FUNDAMENTALS → ADVANCED

# Advanced TAMP Methods & Learning and TAMP

Beyond deterministic, fully-observable, kinematic planning

How does ML factor into TAMP

ICAPS Summer School · Tom Silver & Naman Shah

# Where this session fits

- **Done.** The previous session: TAMP fundamentals
  - abstract vs. concrete domains, task planning, motion planning, downward refinement

- **Now.** How can we reduce the assumptions in TAMP?
- **Next.** Do we need learning? what happens when we add learning?
  - Traditional TAMP works, but is bottlenecked by two things:
    - engineering overhead — a human expert must hand-specify every model
    - planning speed — joint task+motion search is expensive
  - Learning attacks both bottlenecks — and the planner can also supervise learners
- **Claim:** The thesis
  - "TAMP learning" is a distinct area — more than "learning" + "planning" glued together

*Source flow follows: Learning By and For Task and Motion Planning: A Survey (2026).*

# Roadmap

- Advance TAMP methods
- A 60-second TAMP recap — just enough formalism
- Motivation & scope — why combine TAMP and learning
- A taxonomy for TAMP learning — 6 cross-cutting axes
- Family 1 — Learning to make TAMP possible
- Family 2 — Learning to make TAMP fast
- Family 3 — Using TAMP for learning
- The future: opportunities, open challenges, the foundation-model era
- Discussion

*Interactive throughout: a warm-up poll, two think–pair–share breaks, and a hands-on example per family.*

0

# A 60-Second TAMP Recap

Abstract and concrete domains, downward refinement, and the abstract↔concrete interface that learning will target.

# Two domains, linked by abstraction

- **Low level.** Concrete domain
  - states $x \in X$, actions $u \in U$, transitions $f : X \times U \to X$ (continuous, high-dim)
- **High level.** Abstract domain
  - states $s \in S$, actions $a \in A$, transitions $F : S \times A \to S$ (discrete, relational)
- **Two maps.** The glue
  - abstractor $\alpha : X \to S$ (concrete → abstract)
  - concretizer $\gamma : S \to P(X)$ (abstract → set of concrete states / a "mode")
- **A task $\langle x_0, X_g \rangle$ induces an abstract task $\langle s_0, S_g \rangle$ via $\alpha$**

*Planning jointly in both domains beats planning in the concrete domain alone — when the abstraction is good.*

# Downward refinement & the core challenge

- **Downward-refinable planning (the skeleton of every TAMP solver)**
  - 1. find an abstract plan $a_1 \ldots a_T$ reaching S_g
  - 2. refine each abstract step into feasible concrete actions (motions / skill params)
  - 3. backtrack when refinement fails
- **Two refinements add the structure that makes it tractable**
  - relational task planning — objects, predicates, operators → compact, generalizable, heuristics
  - multi-modal motion planning & parameterized skills — modes, kinematic trees, samplers $\delta(\theta|x)$
- **Key.** The hard part: the abstract↔concrete interface
  - abstract plans may NOT be downward-refinable; early choices constrain later feasibility

*This interface is exactly where TAMP learning concentrates its effort.*

# Classic TAMP rests on three convenient assumptions

*The downward-refinable TAMP you just saw assumes a forgiving world. Real robots break all three — and each break has spawned a line of work:*

**1**

**Assumes:** ~~Deterministic transitions~~

**Reality:** actions & outcomes are STOCHASTIC — grasps slip, objects shift

▸ **Stochastic TAMP**
*plans → policies*

**2**

**Assumes:** ~~Full observability~~

**Reality:** perception is PARTIAL — occlusion, uncertain object pose

▸ **Belief-space TAMP**
*perceive–plan–act loop*

**3**

**Assumes:** ~~Skills = kinematic motion~~

**Reality:** real skills are CONTACT-RICH / RL-learned — wiping, opening doors

▸ **Skills + RL in TAMP**
*compose learned skills*

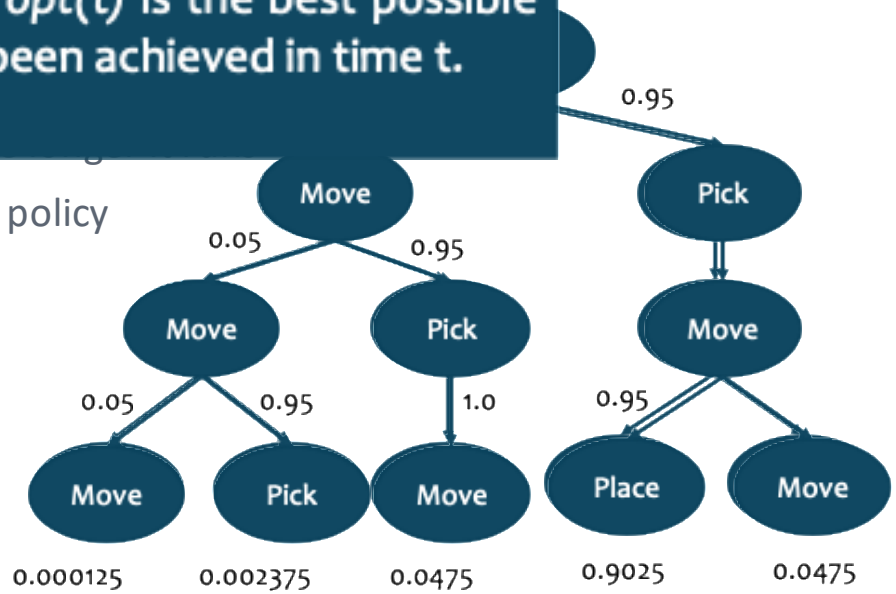
# Anytime Integrated Task and Motion Policies for Stochastic Environments

**Paper.** *Shah, Kala Vasudevan, Kumar, Kamojjhala & Srivastava —*

**Problem.** Actions and the environment are stochastic — a g tion must be a POLICY with a branch for every contingency.

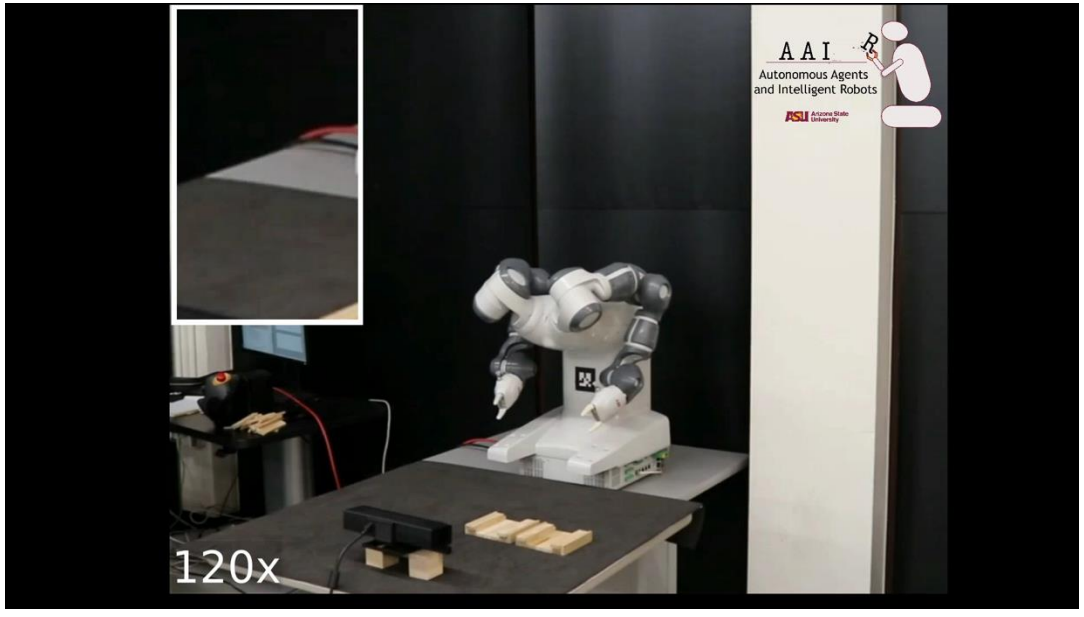
1. **Lift to a stochastic model** cast abstract TAMP
2. **Compute a policy, not a plan** prescribe an act
3. **Refine branches to motions** feed motion-leve
4. **Anytime (ATM-MDP)** probability of hitting an
5. **Guarantee** probabilistically complete; converges toward a hierarchically optimal policy

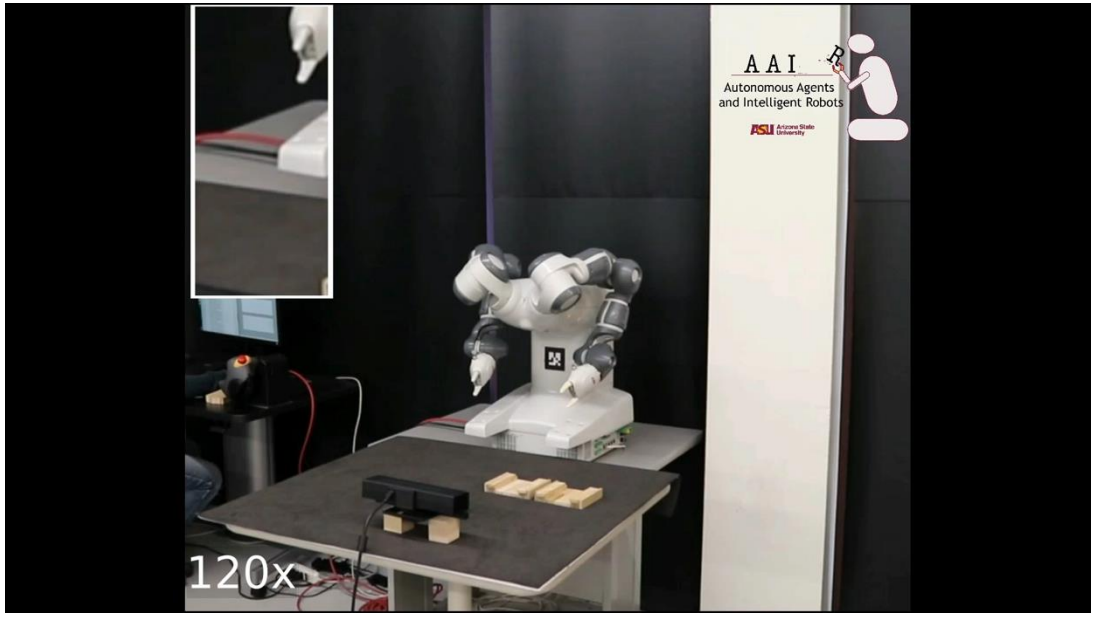
**Theorem:** Let $t$ be the time since the start of the algorithm at which the refinement of any RTL path is completed. If path costs are accurate and constant then the total probability of unrefined paths at time $t$ is at most $1 - opt(t) / 2$, where $opt(t)$ is the best possible refinement that could have been achieved in time $t$.

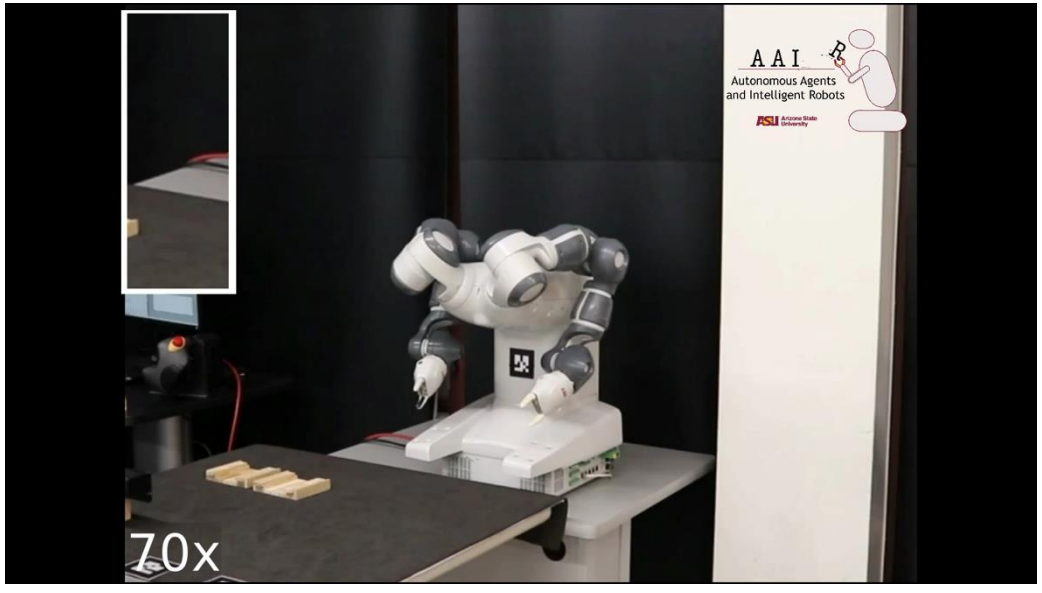


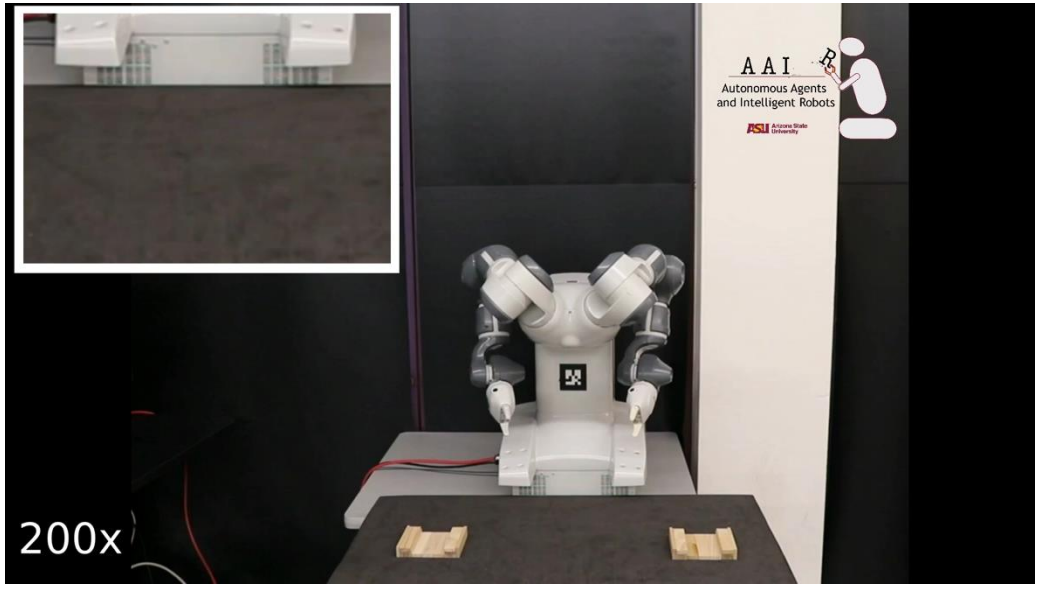

**Takeaway.** From plans to policies: TAMP that expects — and plans for — failure.

# Anytime Integrated Task and Motion Policies for Stochastic Environments



# Online Replanning in Belief Space for Partially Observable TAMP

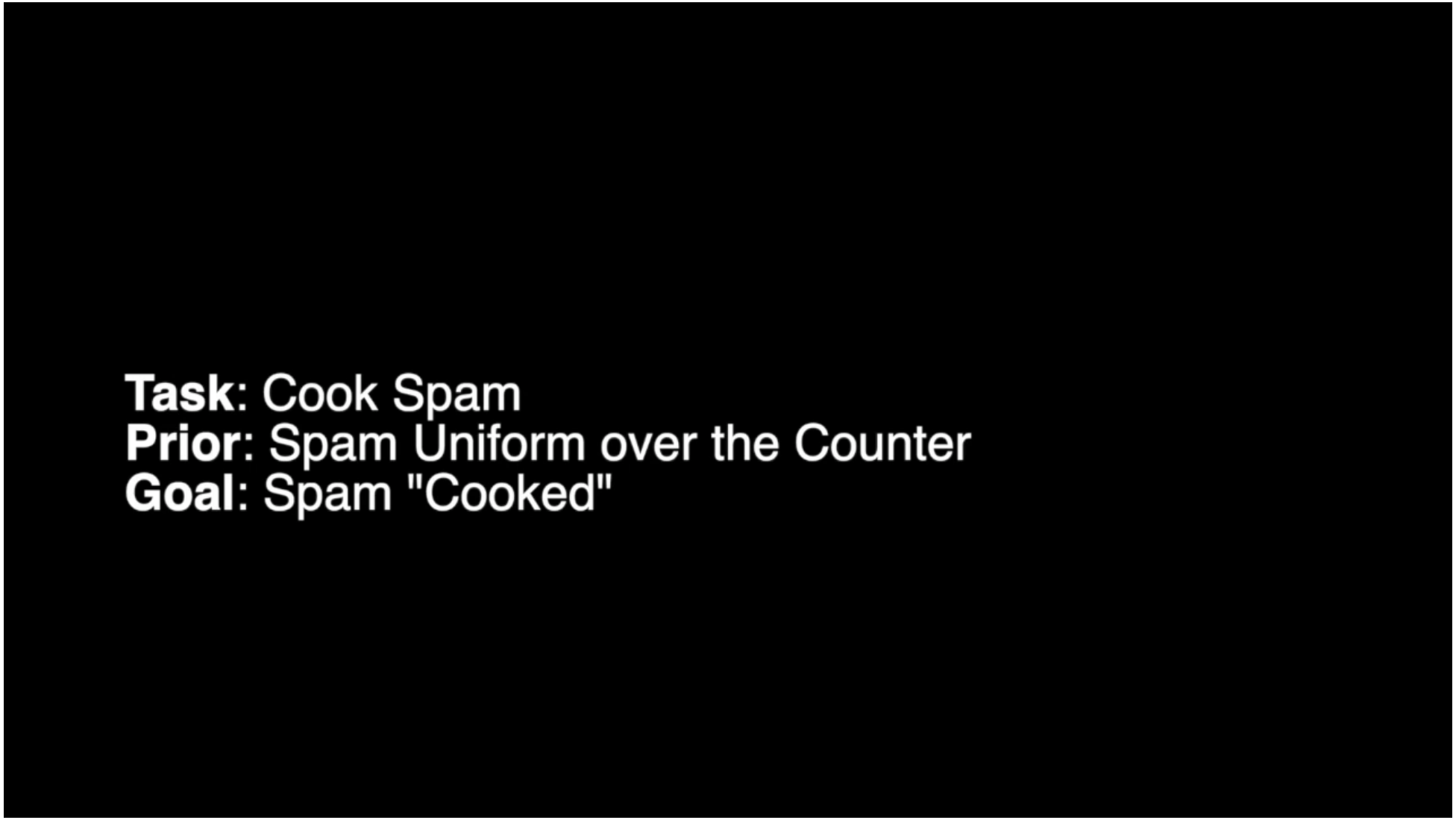
**Paper.** *Garrett, Paxton, Lozano-Pérez, Kaelbling & Fox — ICRA 2020 (SS-Replan / PDDLStream)*

**Problem.** The robot cannot see everything — occlusion and uncertain pose. It must plan over BELIEFS, choose informative observation actions, and revise as the world is revealed.

1. **Plan in hybrid belief space** states carry discrete + continuous uncertainty
2. **Cost-sensitive deterministic planning** pick likely-to-succeed observe + manipulate actions
3. **Execute → observe → update belief** then replan from the new state estimate
4. **Reuse the plan tail** keep the unexecuted suffix → efficient, and never undoes its own progress

**Takeaway.** Closes the perceive–plan–act loop under uncertainty. Validated in simulation and a real-world kitchen.

# Online Replanning in Belief Space for Partially Observable TAMP

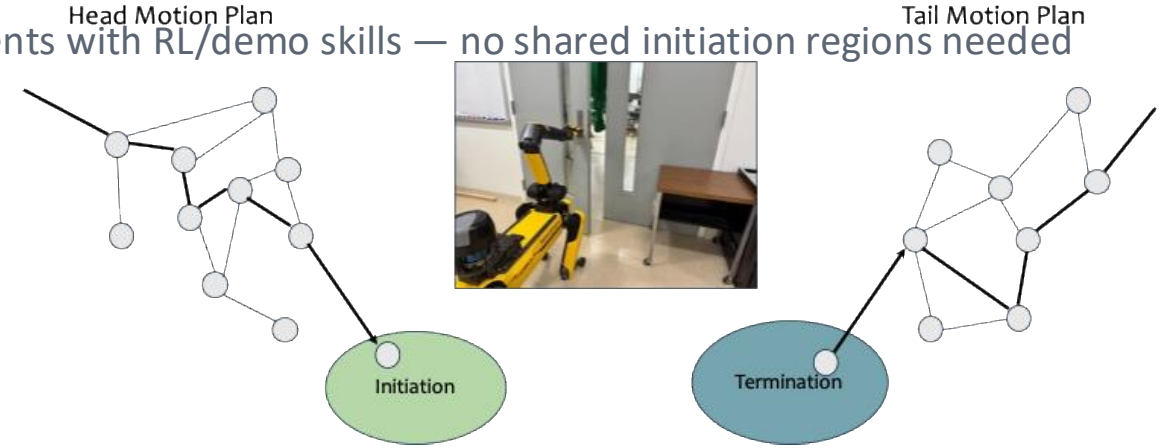


# Beyond Task and Motion Planning: Hierarchical Planning with General-Purpose Policies

**Paper.** *Hedegaard*, Wei*, Yang,, Jaafar, Tellex, Konidaris & Shah — ICRA 2026 (arXiv:2504.17901)*

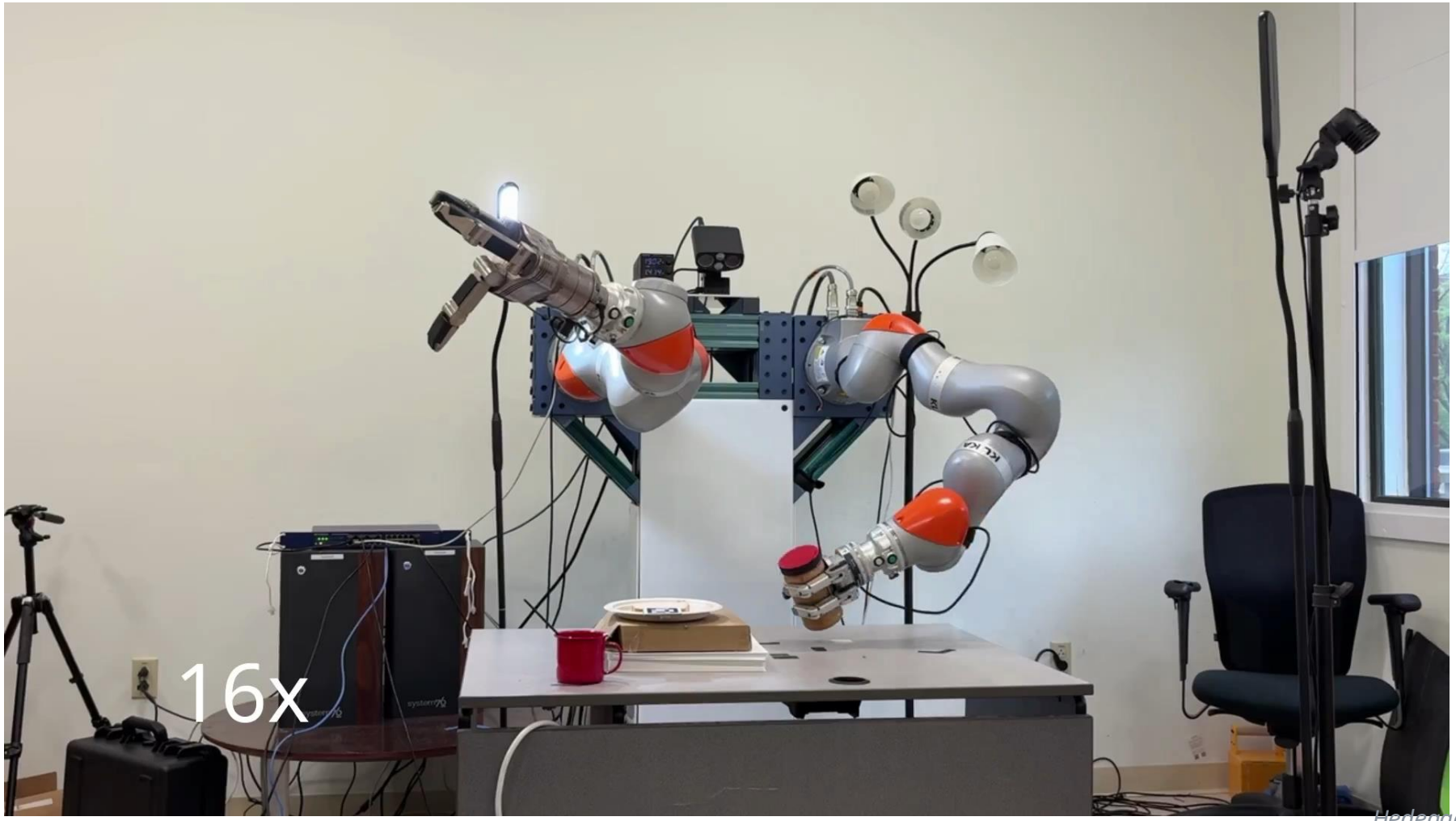
**Problem.** Classical TAMP assumes every skill reduces to kinematic motion planning. But wiping, door-opening, and other contact-rich skills — learned by RL or demonstration — do not. How do we plan with non-kinematic, individually-trained skills?

1. **Composable Interaction Primitive (CIP)** wrap each learned skill: head motion-plan → skill execution → tail motion-plan
2. **Project dynamics to space** CIP "bookends" expose initiation/termination sets a motion planner can connect
3. **Task & Skill Planning (TASP)** each CIP is a symbolic action; a modified ATAM planner sequences them
4. **Compose freely** interleave motion-planned segments with RL/demo skills — no shared initiation regions needed



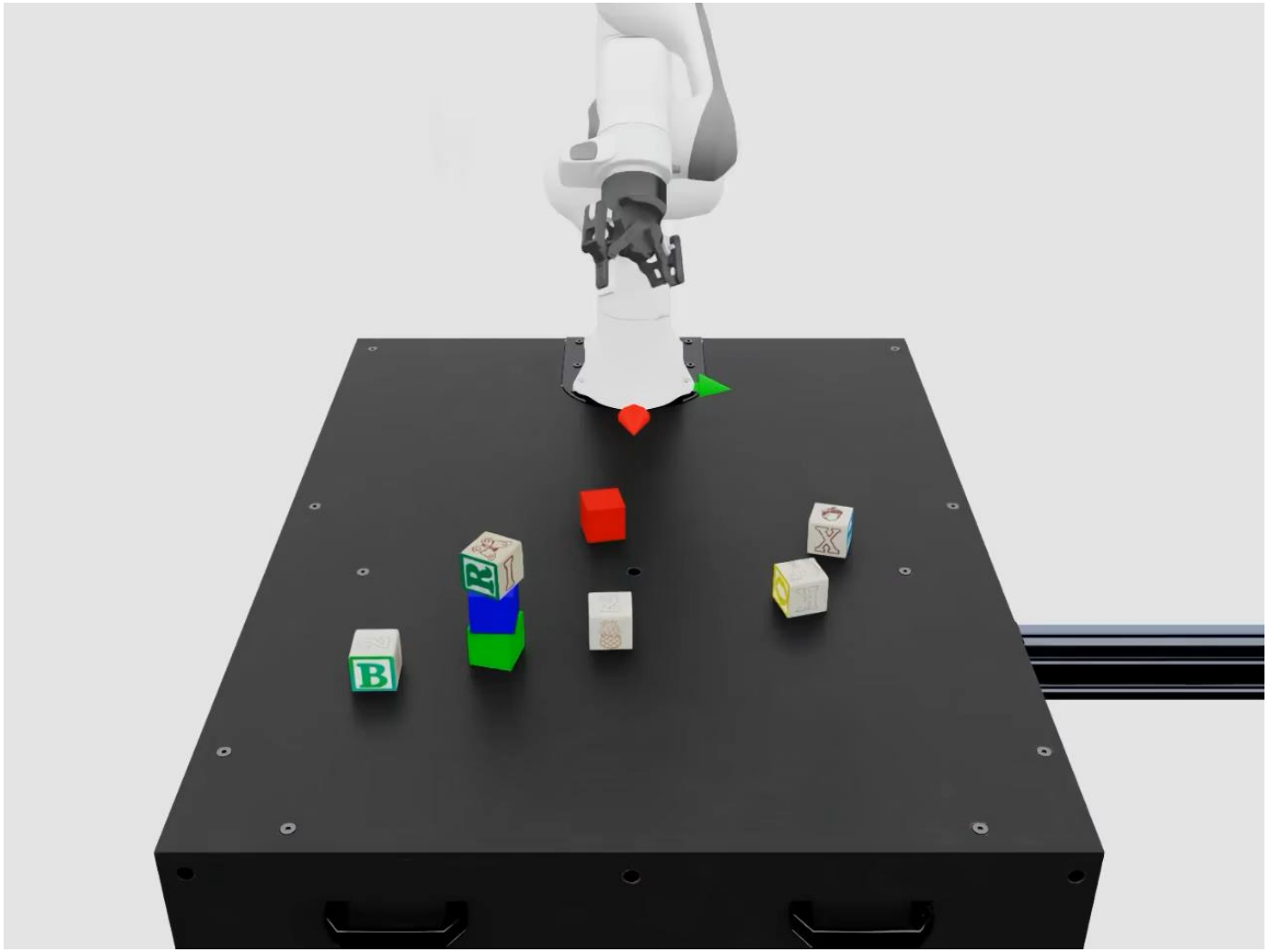

**Takeaway.** Pre-learned RL skills compose inside TAMP. Spot robot: open door → fetch eraser → wipe whiteboard.

# Beyond Task and Motion Planning: Hierarchical Planning with General-Purpose Policies




*Hedegaard*, Wei* et al., 2026*

# Beyond Task and Motion Planning: Hierarchical Planning with General-Purpose Policies



*Shah et al., Ongoing work*

# 1

# Motivation & Scope for TAMP + LEARNING

Robots must be both generalists and specialists. Where does learning help — and what counts as TAMP learning?

# Why TAMP, and why add learning?

- **Robots get semantic goals, not action-by-action instructions**
  - must decide WHAT to do (task planning) and HOW to do it (motion planning), jointly
- **Classical TAMP is powerful but expensive in three currencies**
  - human effort: predicates, operators, skills, samplers, cost models — all hand-built
  - compute: search over task plans × refinement into feasible motions
  - requires privileged and hard to acquire information (e.g., object geometries, poses, etc.)
- **Pure learning (end-to-end policies) struggles where TAMP shines**
  - long horizons, sparse reward, combinatorial structure, constraint satisfaction
- **The opportunity: let each method cover the other's weakness**

*Learning supplies models and speed; TAMP supplies structure and guarantees.*

# 2 A Taxonomy for TAMP Learning

Three families form the spine. Six axes cut across all of them and make design choices consequential.

# The three families — the spine of the field

## 1

### Learning to make TAMP POSSIBLE

Learn the models a human would otherwise hand-build:
predicates, skills, operators, transition models.

**Attacks: engineering overhead**

## 2

### Learning to make TAMP FAST

Use past planning experience to accelerate planning
on harder problems — abstract, concrete, cross-level.

**Attacks: planning speed**

## 3

### Using TAMP FOR LEARNING

Invert the arrow: the planner supervises downstream
learners — distillation, scaffolding, active learning.

**Attacks: data & exploration cost**

*Families 1 & 2 use learning to fix TAMP. Family 3 uses TAMP to fix learning. The grand goal: one agent doing all three over a lifetime.*

# One example we'll carry through all three families

**Setup.** A robot must tidy a cluttered table: stack the blocks, put the mug on the shelf, clear the rest. Long horizon, continuous motions, many objects, sparse reward — the classic TAMP regime.

1. **Family 1 will ask:** where do the symbols On, Holding, Clear and their operators come from?
2. **Family 2 will ask:** with the models in place, how do we plan fast when the table has 30 objects?
3. **Family 3 will ask:** can the planner's solutions teach a policy — or tell us what to label next?

**Takeaway.** Same domain, three lenses. Watch the abstract↔concrete interface in every one.

# Six axes that cut across all three families

| Dimension | Principal split | Qualitative & quantitative trade-offs |
|---|---|---|
| **Data Source** | **External ↔ Planner-generated** | Collection cost · Annotations |
| **Data Selection** | **Given ↔ Planner-selected** | Automation · Coverage |
| **Learning Realm** | **Real ↔ Imagined** | Scalability · Fidelity |
| **Learning Process** | **Decoupled ↔ Interleaved** | Compute · Adaptivity · Stability |
| **Learned Representations** | **Native ↔ Latent** | Interpretability · Reuse · Expressiveness |
| **Formal Guarantees** | **Preserving ↔ Abandoning** | Failure modes · Stakes |

*Locate any paper at the intersection of {family} × {these six axes}. Sparsely-populated cells = research opportunities.*

3

# Learning to Make TAMP Possible

Replace the human engineer. Learn the components of the TAMP system from data: predicates, skills, operators, transition models.

# What must exist for TAMP to run?

- **TAMP needs four components in place — classically all hand-built by an expert:**
  - **Predicates —** the symbolic vocabulary defining abstract states S (On, Holding, Clear)
  - **Actions / operators —** abstract transitions F — preconditions → effects (Pick, Place)
  - **Samplers —** propose continuous parameters $\delta(\theta|x)$ for each action (grasps, placements)
  - **Model —** the transition / dynamics used to refine and check feasibility

- **Three regimes, by how much is given:**
  - predicates given → learn actions & models
  - skills given → learn predicates & operators & models
  - nothing → learn all jointly

*What's given determines what must be learned — and how much classical guarantee you can keep.*

# Regime 1 — Predicates given: learn actions & models

- **Given a predicate vocabulary (On, Holding, …); learn the operators F and their models.**
- **How works do it:**
  - cluster transitions by their lifted effects → one operator per effect set; learn preconditions
  - when no data is given, collect it: goal-literal babbling explores planner-in-the-loop (Chitnis 2021)
  - go further — jointly learn operators + neural samplers + skill policies (neuro-symbolic, Silver 2022)
  - model only what's relevant for planning, not every incidental effect

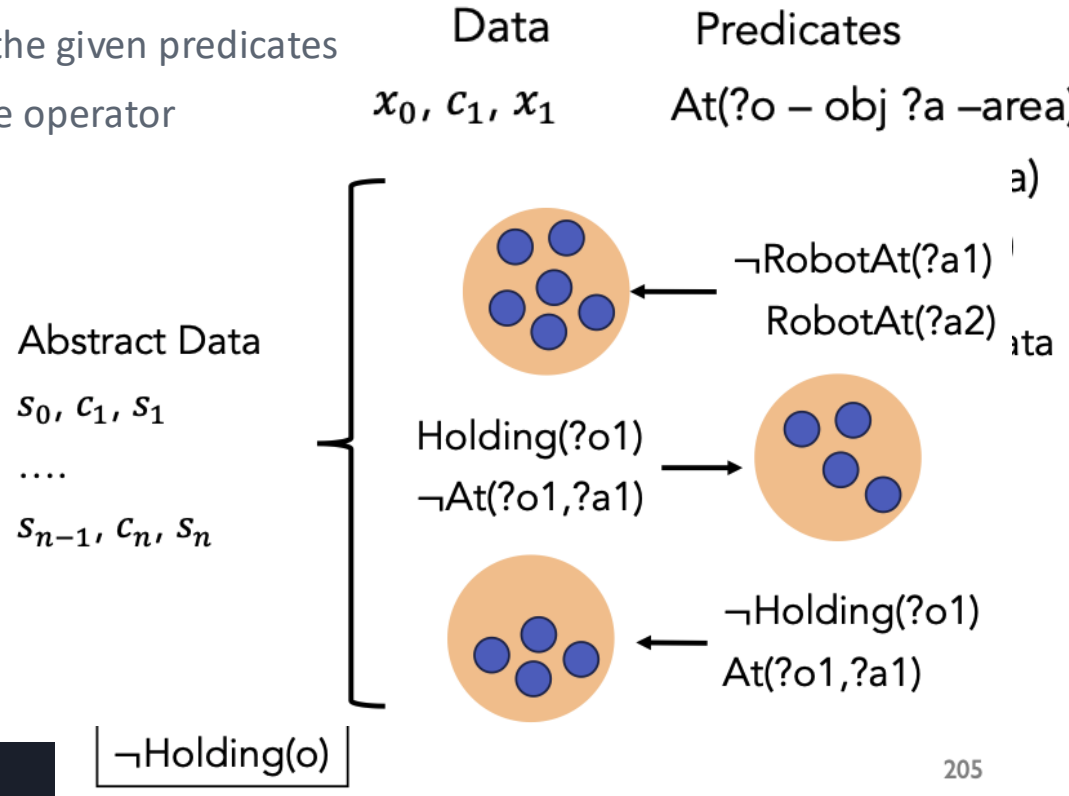
*Predicates fixed; the abstract transition F (with its samplers / model) becomes the learning target.*

PAPER EXAMPLE

# Learning Symbolic Operators for Task and Motion Planning

**Paper.** *Silver, Chitnis, Tenenbaum, Lozano-Pérez & Kaelbling — IROS 2021*
**Setup.** Predicates are given; learn the OPERATOR set (the abstract transition F) directly from transition data.

1. **Collect transitions** and compute lifted effects — add/delete sets over the given predicates
2. **Cluster by effect set** each distinct effect cluster becomes one candidate operator
3. **Learn preconditions** what must hold for that effect to occur
4. **Plan** use the learned operators in a downward-refinable TAMP loop




**Takeaway.** The canonical "operators given predicates" recipe — cluster effects, then learn preconditions.

# Regime 2 — Skills given: learn predicates

- **Given a fixed set of skills (initiation sets + effects); learn the symbols needed to plan with them.**
- **Key idea — predicates as the EFFECTS of skills:**
  - the symbols a planner must track = each skill's preconditions and its image (sets of states)
  - construct the coarsest abstraction that suffices to evaluate any plan over those skills
  - the abstraction is DERIVED from skills — not invented for convenience
  - modern twist: a VLM proposes a new predicate when a skill's outcome is ambiguous (Athaye et al., 2025, Yang et al., 2026)

*Skills fixed; the abstraction is derived from them — and can be made provably sound & complete.*

PAPER EXAMPLE

# From Skills to Symbols

**Paper.** *Konidaris, Kaelbling & Lozano-Pérez — JAIR 2018*
**Setup.** Reverse the usual order: given a fixed set of skills, DERIVE the symbolic vocabulary a planner needs.

1. **Each skill has structure** an initiation set (where it can run) and effects (where it lands)
2. **Symbols = those sets** the planner tracks skill preconditions and images as symbols
3. **Construct coarsest abstraction** just fine enough to evaluate any plan over the given skills
4. **Result** a propositional PDDL-like model — provably SOUND and COMPLETE for those skills

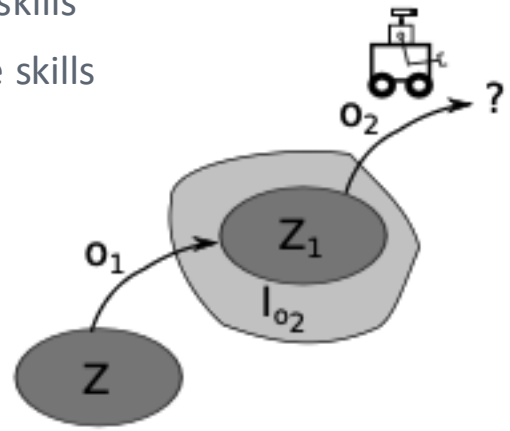
**Takeaway.** The near-singular guarantee-PRESERVING construction. Symbols are an emergent consequence of skills.

PAPER EXAMPLE

# From Skills to Symbols

**Paper.** *Konidaris, Kaelbling & Lozano-Pérez — JAIR 2018*
**Setup.** Reverse the usual order: given a fixed set of skills, DERIVE the symbolic vocabulary a planner needs.

1. **Each skill has structure** an initiation set (where it can run) and effects (where it lands)
2. **Symbols = those sets** the planner tracks skill preconditions and images as symbols
3. **Construct coarsest abstraction** just fine enough to evaluate any plan over the given skills
4. **Result** a propositional PDDL-like model — provably SOUND and COMPLETE for those skills




**Takeaway.** The near-singular guarantee-PRESERVING construction. Symbols are an emergent consequence of skills.

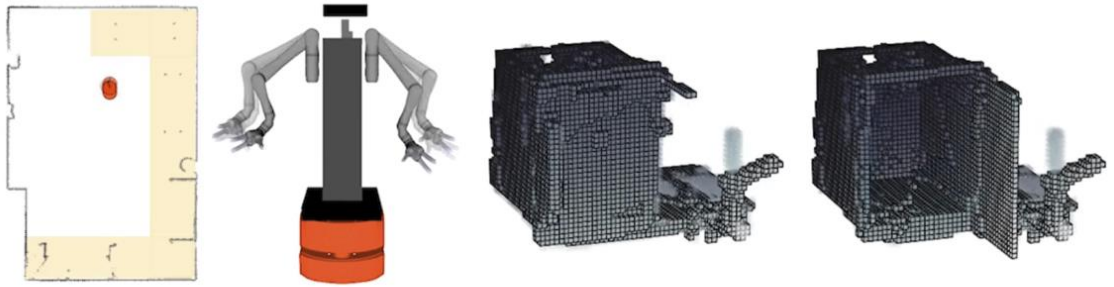
PAPER EXAMPLE

# From Skills to Symbols

**Paper.** *Konidaris, Kaelbling & Lozano-Pérez — JAIR 2018*

**Setup.** Reverse the usual order: given a fixed set of skills, DERIVE the symbolic vocabulary a planner needs.

```
(:action cupboard_open1
 :parameters ()
 :precondition (and (symbol1) (symbol3) (symbol4))
 :effect       (and (symbol5) (not (symbol4))
                    (decrease (reward) 67.44))
)
```



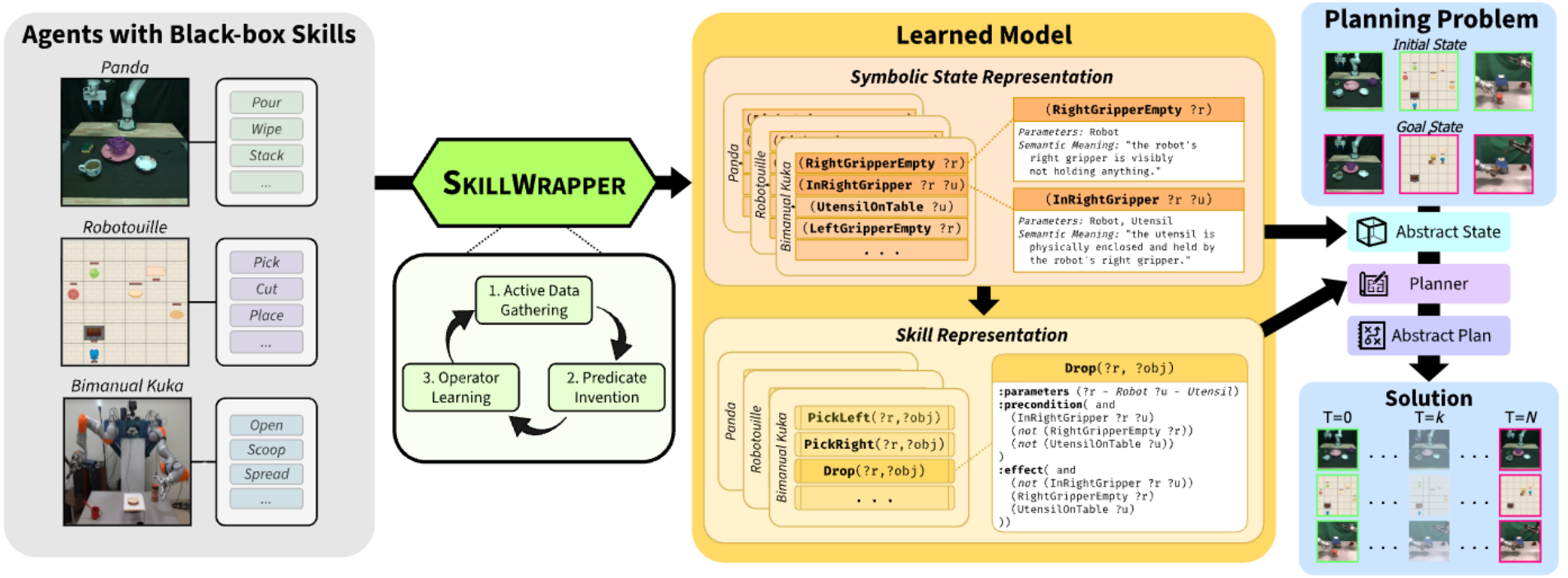

**Takeaway.** The near-singular guarantee-PRESERVING construction. Symbols are an emergent consequence of skills.

PAPER EXAMPLE

# SkillWrapper

**Paper.** *Yang, Hedegaraard, …, Shah, 2026*

**Setup.** Reverse the usual order: given a fixed set of skills, DERIVE the symbolic vocabulary a planner needs – Now using a VLM



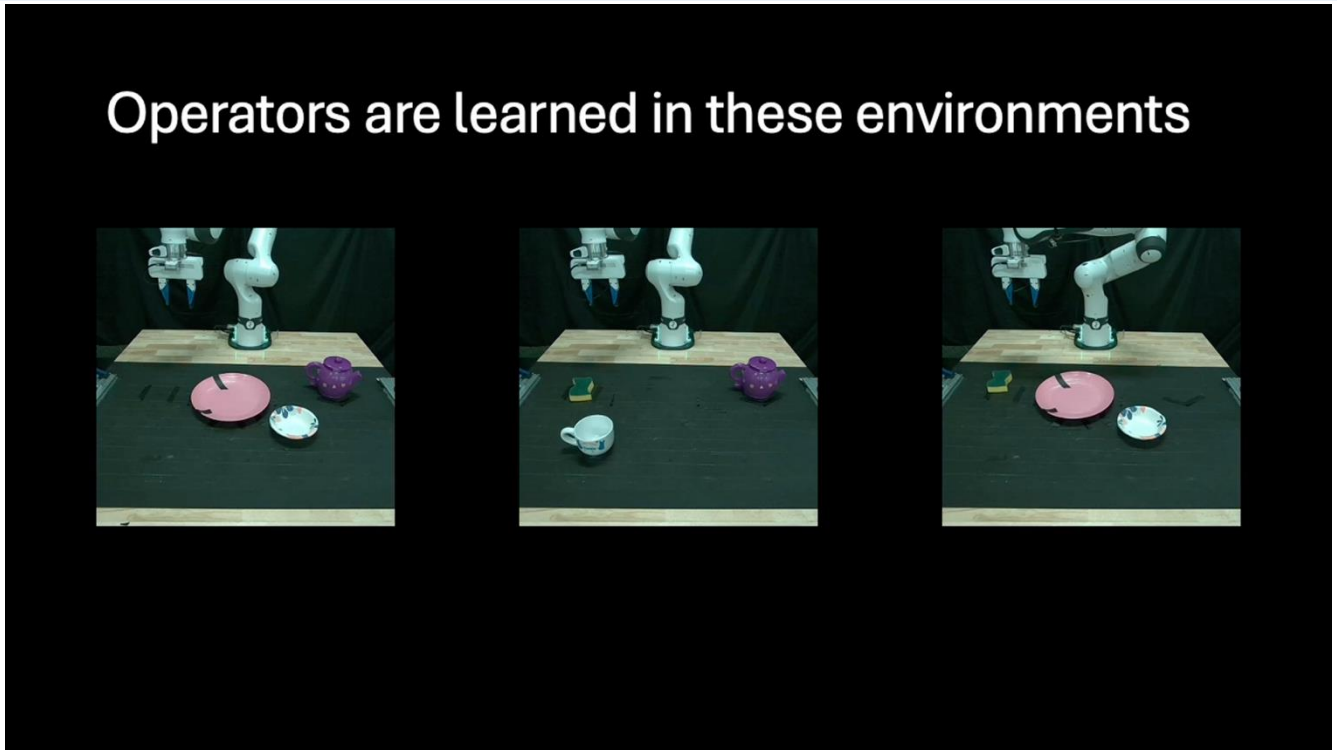

**Takeaway.** The near-singular guarantee-PRESERVING construction. Symbols are an emergent consequence of skills.

PAPER EXAMPLE

# SkillWrapper

**Paper.** *Yang, Hedegaraard, …, Shah, 2026*

**Setup.** Reverse the usual order: given a fixed set of skills, DERIVE the symbolic vocabulary a planner needs – Now using a VLM




**Takeaway.** The near-singular guarantee-PRESERVING construction. Symbols are an emergent consequence of skills.

# Regime 3 — Nothing given: jointly learn everything

- **Given only data — unannotated robot trajectories or demonstrations.**
- **Learn the entire TAMP domain bottom-up, all four components at once:**
  - invent predicates that partition the continuous space into planning-relevant regions
  - invent abstract actions and their precondition/effect models (a PDDL-like domain)
  - learn samplers for the continuous parameters; learn or use a transition model to refine
  - related: invent predicates from a grammar, kept for planning utility (Predicate Invention, Silver 2023)

*The hardest, most autonomous regime — and the one that usually trades away classical guarantees.*

PAPER EXAMPLE

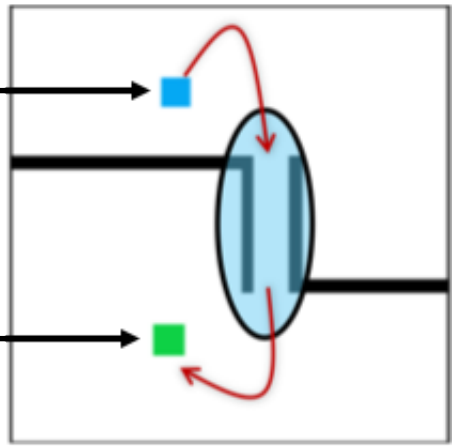
# From Reals to Logic and Back

**Paper.** *Shah, Nagpal, Verma & Srivastava — CoRL 2025*

**Setup.** Invent an ENTIRE PDDL domain — predicates, actions, samplers, and models — from unannotated, high-dimensional, real-valued robot trajectories. No human-given symbols.

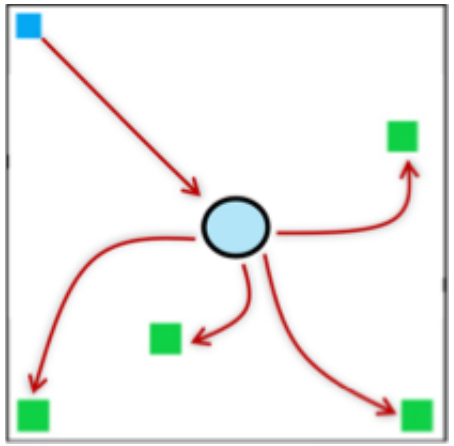
Given a class of motion planning problem M, criticality of an open set $r$ in the C-space:

$$\mu(r) = \lim_{s_n \to^+ v} \frac{f(r)}{v(s_n)}$$




$f(x)$ = fraction of solution plans that pass through $x$ // captures hubs
$v(x)$ = measure of $x$ under uniform sampling density //captures bottlenecks.

[Molina et al., 2020, ICRA]



PAPER EXAMPLE

# From Reals to Logic and Back

**Paper.** *Shah, Nagpal, Verma & Srivastava — CoRL 2025*

**Setup.** Invent an ENTIRE PDDL domain — predicates, actions, samplers, and models — from unannotated, high-dimensional, real-valued robot trajectories. No human-given symbols.

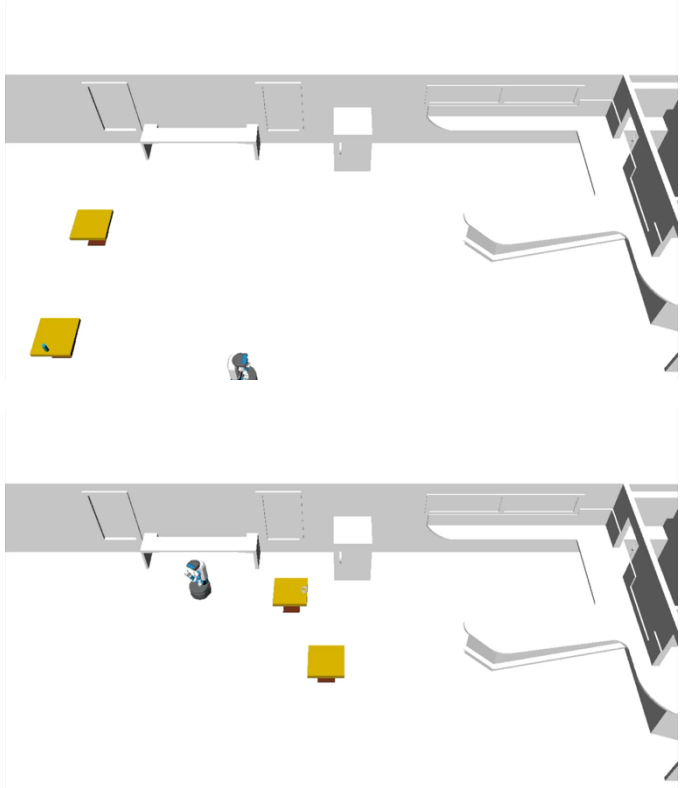
1. **Input** raw real-valued trajectories, no labels or predefined vocabulary
2. **Invent state predicates** that partition continuous space into planning-relevant regions
3. **Invent actions + models** learn symbolic preconditions/effects and samplers → a PDDL-like domain
4. **Plan & refine** off-the-shelf task planner over invented symbols, then motion refinement

PAPER EXAMPLE

# From Reals to Logic and Back

**Paper.** *Shah, Nagpal, Verma & Srivastava — CoRL 2025*

**Setup.** Invent an ENTIRE PDDL domain — predicates, actions, samplers, and models — from unannotated, high-dimensional, real-valued robot trajectories. No human-given symbols.



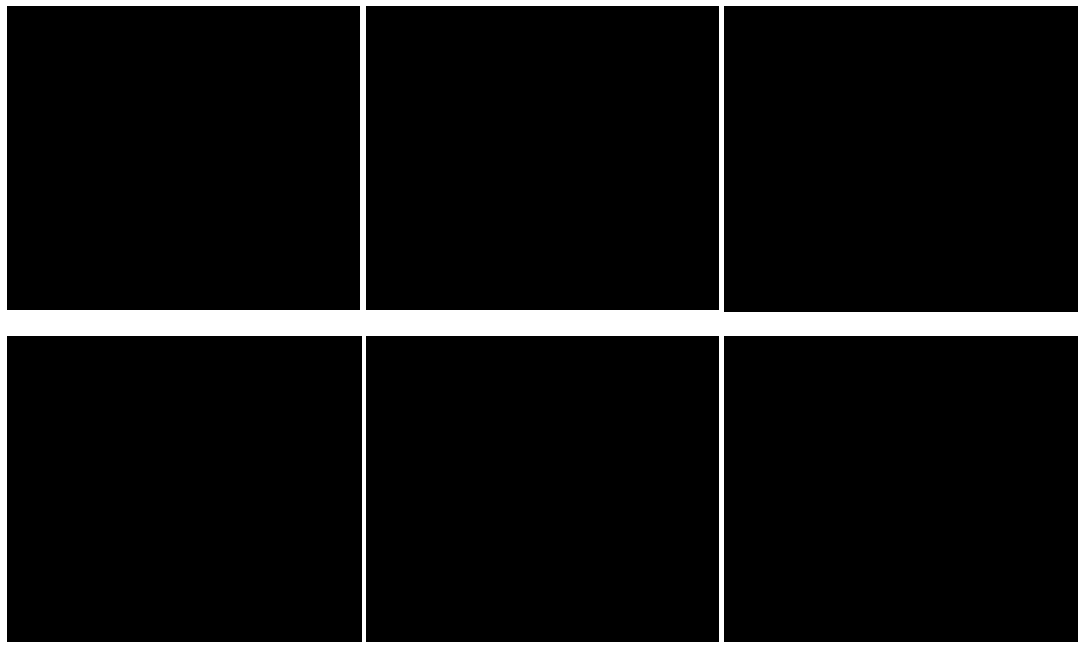
**Takeaway.** The far end of "make TAMP possible": the whole abstraction, bottom-up from data. Domains: Keva, cafe delivery, can-packing.

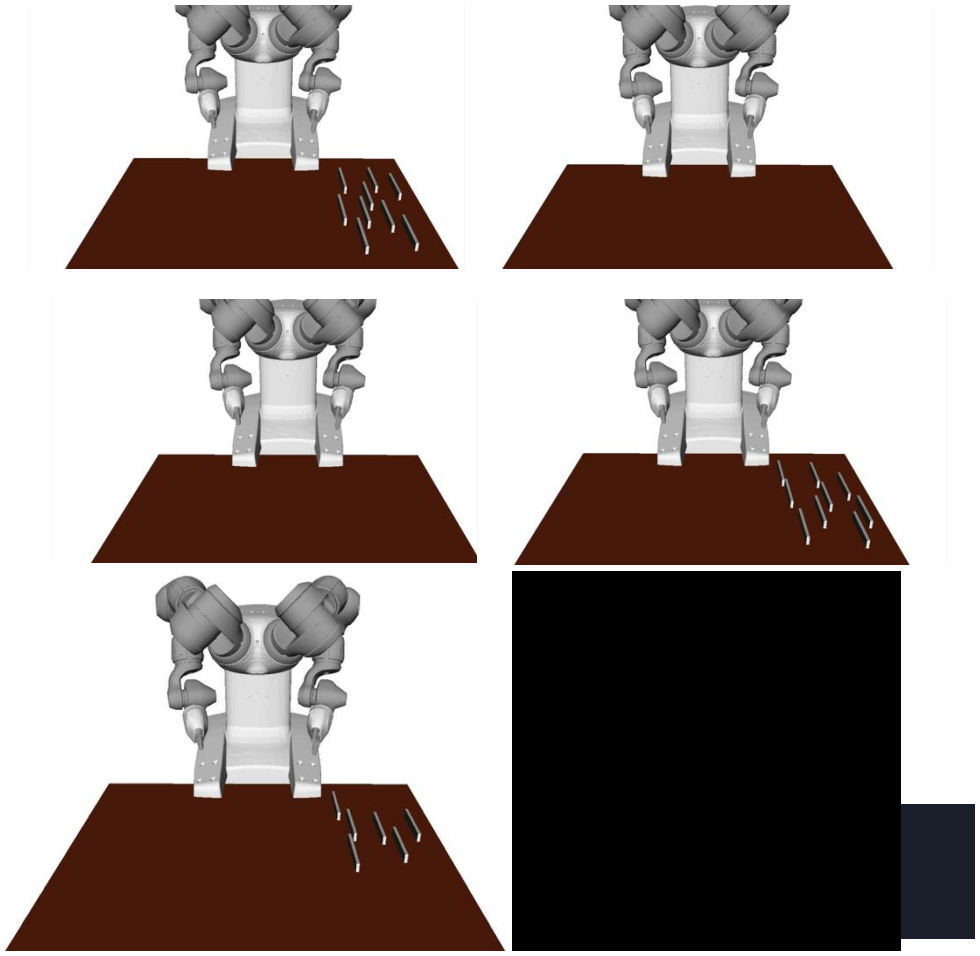
PAPER EXAMPLE

# From Reals to Logic and Back

**Paper.** *Shah, Nagpal, Verma & Srivastava — CoRL 2025*

**Setup.** Invent an ENTIRE PDDL domain — predicates, actions, samplers, and models — from unannotated, high-dimensional, real-valued robot trajectories. No human-given symbols.





**Takeaway.** The far end of "make TAMP possible": the whole abstraction, bottom-up f

# Inventing the abstraction for the table

**Setup.** No predicates or operators are given — only a handful of demonstrations of a human tidying the table. We want a symbolic vocabulary the planner can use.

1. **Propose predicates** from demos / a VLM: candidates like On(a,b), Holding(a), Clear(a), GraspFree()
2. **Keep what helps** retain a predicate only if it improves planning on held-out tidying tasks (instrument view)
3. **Cluster into operators** group transitions by lifted effects → Pick, Place schemas with preconditions
4. **Attach samplers** neuro-symbolic: each operator gets a learned sampler $\delta(\theta|x)$ for grasps/placements
5. **Plan & refine** the invented $\alpha$, $\Sigma$, F now drive downward-refinable planning on a fresh, cluttered table

**Takeaway.** This is the abstraction-learning

✦ DISCUSS · 2 min

# Predict the failure mode

*Call out a failure, then name which learned component you'd fix.*

- You invented predicates that perfectly predict the demos. The planner still fails on the real table. Why?
- Hint 1: lossy abstraction — many concrete states map to one symbol
- Hint 2: a grasp chosen for Pick can make the later Place infeasible (the interface problem)

4

# Learning to Make TAMP Fast

All models are in place, but planning is slow. Use experience on easy problems to accelerate hard ones — organized by where the model operates.

# Where can a learned model accelerate planning?

- **Organizing axis: position relative to the abstract/concrete boundary**
  - Abstract acceleration— speed up the symbolic planner
  - Concrete acceleration — speed up refinement / motion planning
  - Cross-level acceleration— manage the interaction between the two
- **Most experience is intrinsic: collected by running a planner on easier problems**
- **Key contrast.** Watch the guarantees axis throughout:
  - re-ordering search → preserves completeness · pruning search → abandons it

*Same model can fall in several buckets; classified by its primary contribution.*

# Abstract acceleration — three sub-families

- **Bias or bypass search.** Abstract policies
  - GNN priority over skeletons (Khodeir 2023); LLM prior warm-starts skeleton search (Lee 2025)
  - bypass: image+goal → abstract action sequence, then refine (Driess 2021)
- **Shorter sub-problems.** Subgoal decomposition
  - TAMP time grows super-linearly in plan length → decomposition gives large speedups + local replanning
  - hand sketches · LLM/VLM subgoals at inference · learned from demos (GNN matches mined subgoals)
- **Shrink the problem.** Task reduction
  - score object relevance with a GNN, plan over the high-scoring subset, fall back if it fails (Silver 2021)
  - LLM semantic search over a 3D scene graph → task-relevant subgraph (Rana 2023)

# Concrete acceleration

## Concrete acceleration — make refinement cheaper

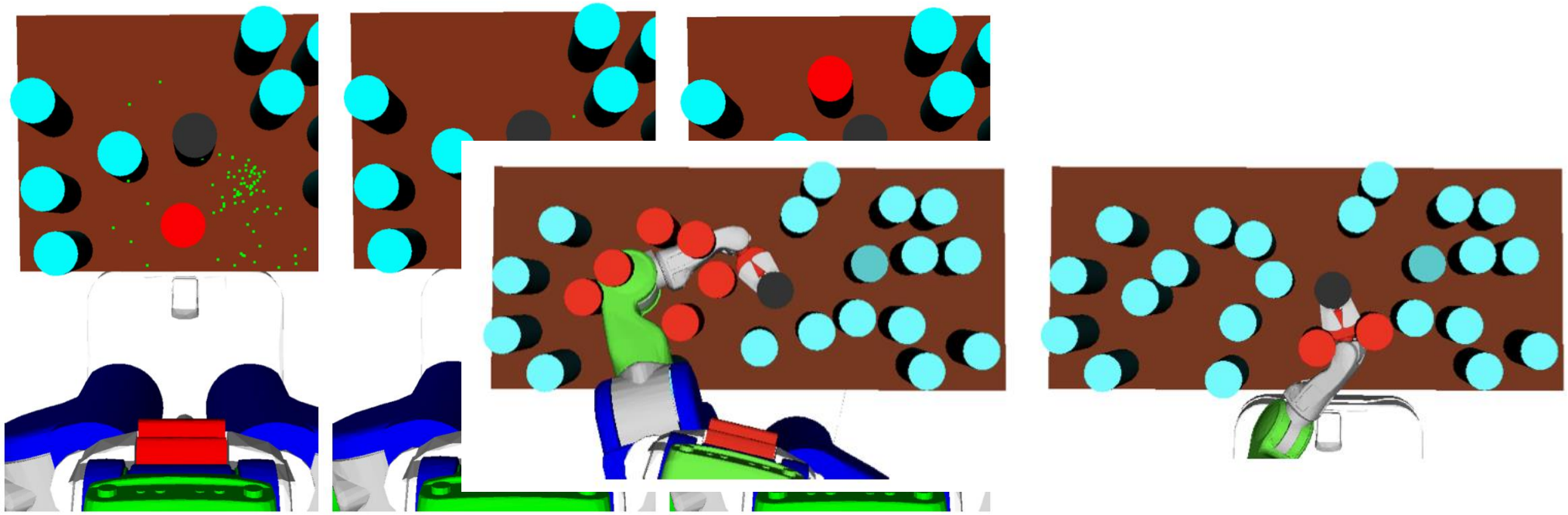
- learned samplers $\delta(\theta|x)$: propose feasible grasps/placements instead of rejection sampling
- learned feasibility / reachability predictors: prune infeasible refinements before motion planning
- warm-start or replace motion planning with learned trajectory predictors

*Concrete = cheaper feasibility checks; cross-level = smarter allocation of effort between layers.*

PAPER EXAMPLE

# Learning Sampler Heuristics

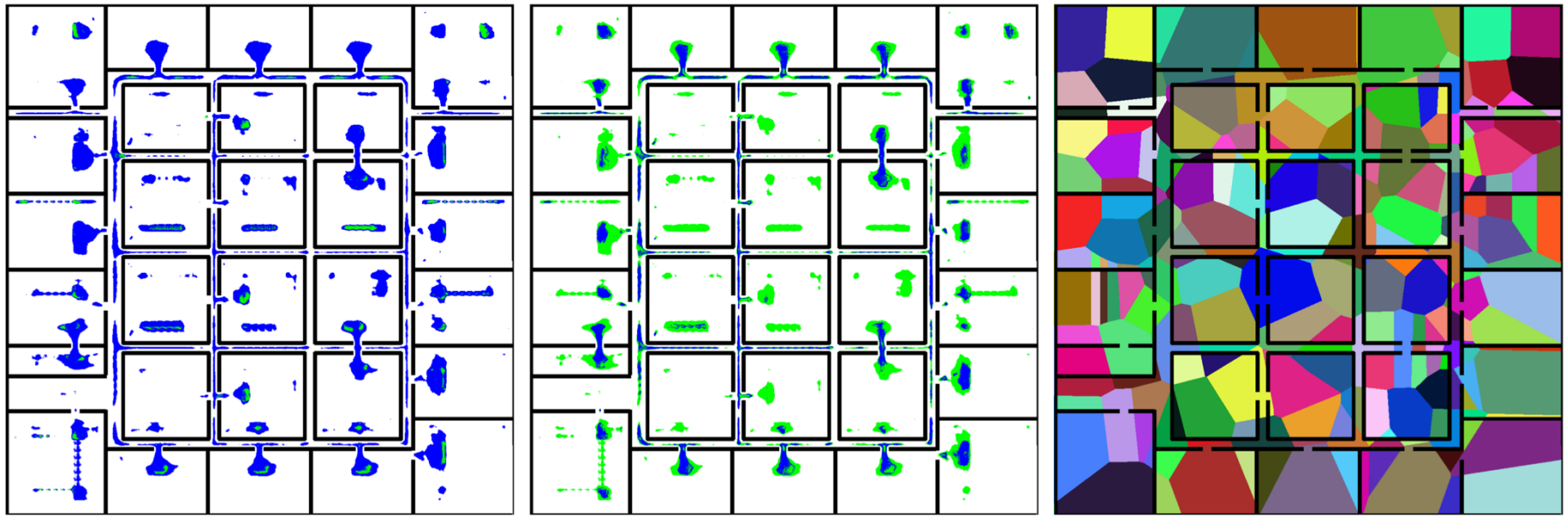
**Paper.** *Chitnis, Hadfied-Menell, Gupta, Srivastava, Groshev, Lin, Abbeel, 2016*
**Setup.** Guided Search for Task and Motion Plans Using Learned Heurstics



PAPER EXAMPLE

# HARP: Speeding Up the Motion Planning

**Paper.** *Chitnis, Hadfied-Menell, Gupta, Srivastava, Groshev, Lin, Abbeel, 2016*
**Setup.** Using Deep Learning to Bootstrap Abstractions for Hierarchical Robot Planning



# Cross-level acceleration

**Cross-level acceleration — manage the abstract↔concrete loop**

- learn which task plans are worth refining (refinement failures feed back into the choice)
- online priority updates: a skeleton that fails refinement is down-weighted (Khodeir 2023)
- this is where the central TAMP difficulty — choosing what to refine — becomes a learning problem

*Concrete = cheaper feasibility checks; cross-level = smarter allocation of effort between layers.*

# Same table, now with 30 objects

**Setup.** The invented abstraction from Family 1 works, but the table is now crowded. Naive TAMP times out: the task planner drowns in objects and refinement keeps sampling infeasible grasps.

1. **Reduce the problem** a GNN scores object relevance → plan over the ~6 objects that matter, fall back if needed
2. **Decompose the horizon** split "tidy everything" into per-region subgoals; replan locally on failure
3. **Accelerate refinement** the learned samplers $\delta(\theta|x)$ now also rank grasps by feasibility, cutting rejections
4. **Close the loop** skeletons that fail refinement get down-weighted online — cross-level learning
5. **Mind the guarantee** re-ordering search keeps completeness; pruning objects abandons it — choose deliberately

**Takeaway.** None of this changes WHAT is planned — only HOW fast the same abstraction is searched.

5

# Using TAMP for Learning

Invert the arrow. Robot learning is bottlenecked by data, exploration, and missing long-horizon structure — let the planner supply all three.

# The inversion: planner as a source of supervision

- **Robot learning's bottlenecks…**
  - demonstration data is costly · RL exploration is expensive · long-horizon structure is hard to discover
- **…are exactly what a planner can provide**
  - **remove TAMP at runtime —** Distillation
    - train a standalone learner to imitate/replace the pipeline; planner present only at training
  - **keep TAMP at deployment —** Scaffolding
    - use learning to extend it: new skills, recovery policies, learned primitives
  - **spend data wisely —** Active learning
    - use planning to allocate scarce data collection to the components that need it

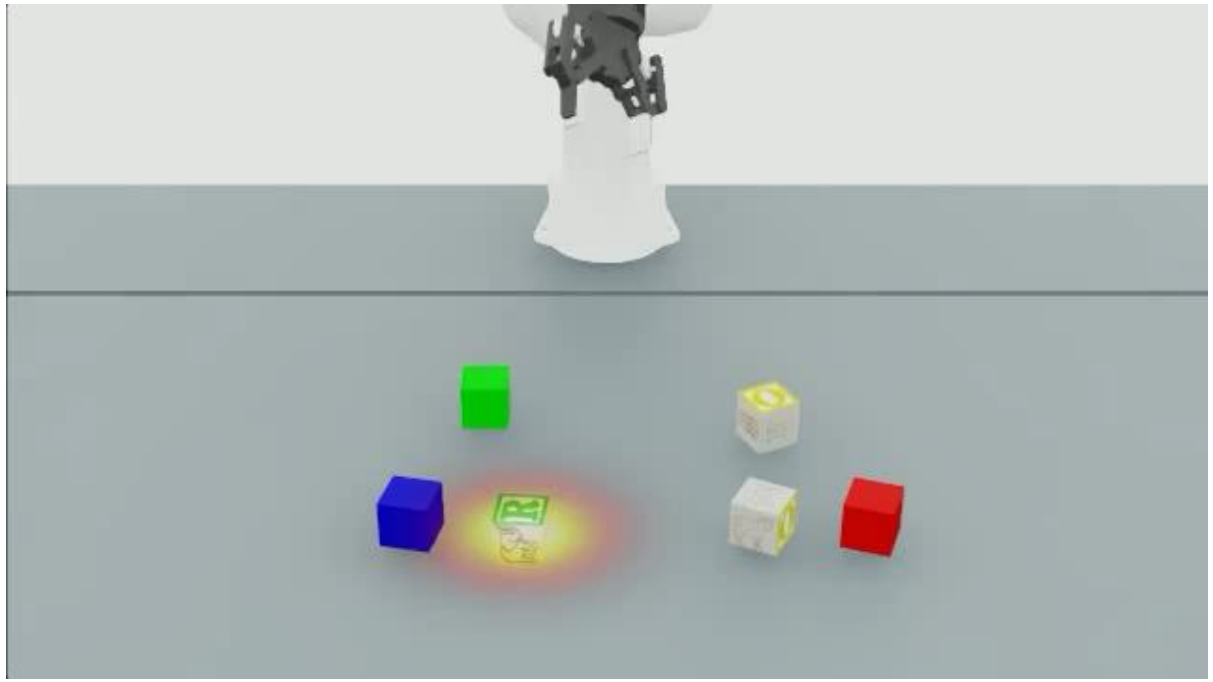
*Distinguished by what the learner inherits from TAMP and whether TAMP survives to deployment.*

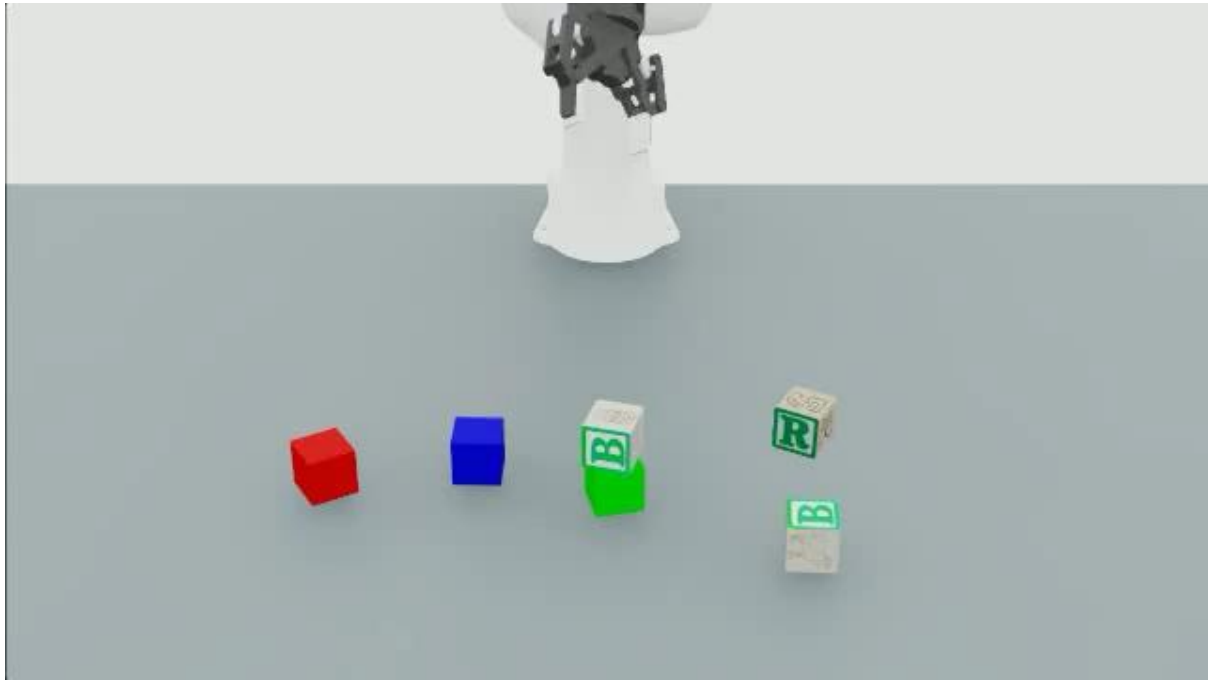
# Distillation — three flavors

- **Out-of-the-loop by construction: planner generates data, learner trains separately**
  - **Keep the 2 levels.** Hierarchical distillation
    - learn a high-level policy + operator-specific controllers; task plans supervise top, motions the bottom
  - **Flatten it.** End-to-end policy distillation
    - single sensor→action policy via BC on planner trajectories; curate to reduce multi-modality (Dalal 2023)
    - distill Skill-RRT or model-based planners into diffusion / transformer policies (Jung 2025, Li 2024)
  - **Into RL.** Distillation via reward shaping
    - convert a symbolic plan into a potential function → shaped reward that accelerates RL (Grzes 2008+)

*Trade-off: flattening removes runtime planning but discards structure — see "native representations" opportunity.*

# Distillation — three flavors





**Flat Policy**

*Trade-off: flattening removes runtime planning but discards structure — see "native representations" opportunity.*

# Scaffolding

- **Scaffolding — TAMP stays at deployment, learning extends it**
  - learn new skills / action primitives the engineered library lacks
  - learn recovery and bridge policies for situations the planner can't handle (novelty, failures)
  - the planner provides the long-horizon backbone; learners fill local gaps

*Scaffolding = capability extension; active learning = data-efficiency via planning-guided querying.*

# Active learning

- **Active learning — plan to decide what data to collect**
  - use the planner to identify which component, if improved, most helps task success
  - allocate scarce human effort: atom labels, demonstrations, queries (Vats et al. 2025)
  - closes a loop: plan → find the weak component → collect targeted data → re-learn

*Scaffolding = capability extension; active learning = data-efficiency via planning-guided querying.*

# Letting the table-tidying planner teach

**Setup.** We can now solve the table reliably — but slowly, and only with accurate state estimation. We'd like a fast policy, and we have limited human labeling budget.

1. **Distill** run TAMP offline to generate thousands of tidying trajectories → behavior-clone a policy
2. **Keep structure** distill hierarchically (high-level over operators + per-operator controllers), or flat
3. **Go native (open problem)** distill into a generalized plan / program over On, Holding — auditable, editable
4. **Scaffold** where the policy fails (a novel object), fall back to TAMP and learn a bridge skill
5. **Actively label** use planning to find which atom (e.g. is Clear true?) most needs a human label next

**Takeaway.** The planner pays its abstraction forward — as data, as structure, and as a labeling oracle.

# Distill to what?

*Pair up, fill the gain/lose table, then we collect answers on the board.*

- You can distill your table-tidying planner into (a) a neural policy, or (b) a program / generalized plan
- What do you GAIN and LOSE with each, on: speed · interpretability · reuse · safety?
- When would you refuse to distill at all and keep the planner in the loop?

6

# The Future of TAMP Learning

Clear opportunities the taxonomy exposes, harder open challenges, and why TAMP learning matters more — not less — in the foundation-model era.

# Clear opportunities — thinly-populated cells

- **#1.** Guarantee-preserving learning
  - methods almost all abandon soundness/completeness (Konidaris a near-singular exception)
  - extend preservation to learned MODELS — e.g. skills with bounded-competence estimates
- **#2.** Native representations in distillation
  - distillation is entirely latent today → distill into programs, rules, generalized plans (auditable, editable)
- **#3.** In-the-loop acceleration learning
  - learning to make TAMP fast is largely offline → adaptively select instances that reveal where planning will fail
- **#4 — the holy grail.** Cross-family integration
  - one agent that accumulates components, accelerates its own planning, and teaches learners

# Open challenges — harder, unsettled

- **Robustness under model error**
  - TAMP is brittle to geometry/friction/perception error; learned components add more error
  - need graceful degradation: bounded-uncertainty planning, fallback hierarchies, contract-based composition
- **Partial observability & active perception**
  - most of the field assumes accurate state estimation; real deployments don't get it
  - learn when to look, where to look, what to model — barely exploited
- **Humans beyond demonstrations**
  - input is dominated by demos/labels/prompts; preferences, corrections, intent, mixed-initiative are underused

*These cut across all three families and lack settled answers.*

# TAMP learning in the foundation-model era

- **Does TAMP learning still matter as FMs get better? → Yes, more so.**
- **Non-overlapping competences.** Complementary strengths
  - FMs: open-vocab perception, common sense, semantic generalization, language interfaces
  - TAMP: long-horizon constrained reasoning, certified completeness, domains with no pretraining data
- **Across all three families.** FMs are now supplying TAMP components
  - predicates proposed by VLMs · plan skeletons & decompositions from LLMs · skills bootstrapped by FMs
  - e.g. TipTop couples a GPU-parallel TAMP solver with pretrained perception/grounding/grasping modules
- **The bet: FMs fill TAMP's classic input assumptions; TAMP supplies the reasoning FMs lack**

# Takeaways

- TAMP learning is a distinct area — more than "planning" + "learning"
- Three families:  make TAMP possible · make TAMP fast · use TAMP for learning
- Six axes locate any method:  source · selection · realm · process · representation · guarantees
- The guarantees axis is the field's biggest gap — most learning abandons soundness/completeness
- The holy grail is cross-family integration: one agent improving on all three axes over its lifetime
- Foundation models populate the taxonomy — they don't dissolve it

DISCUSSION

- Which axis should the field prioritize — guarantees, robustness, or perception?
- When is learning a TAMP component worth losing its guarantees?
- What should be distilled into native, auditable representations — and what shouldn't?
- Where do foundation models help, and where do they quietly break things?

## Thank you — questions & discussion

*Reference: "Learning By and For Task and Motion Planning: A Survey" (2026)*