Linköping University

Automated Planning

The Partial Order Causal Link Search Space

Jonas Kvarnström
Department of Computer and Information Science
Linköping University

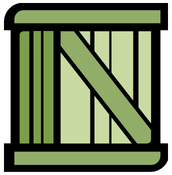
Partly adapted from slides by Dana Nau

jonas.kvarnstrom@liu.se – 2019

Motivations

Motivating Problem

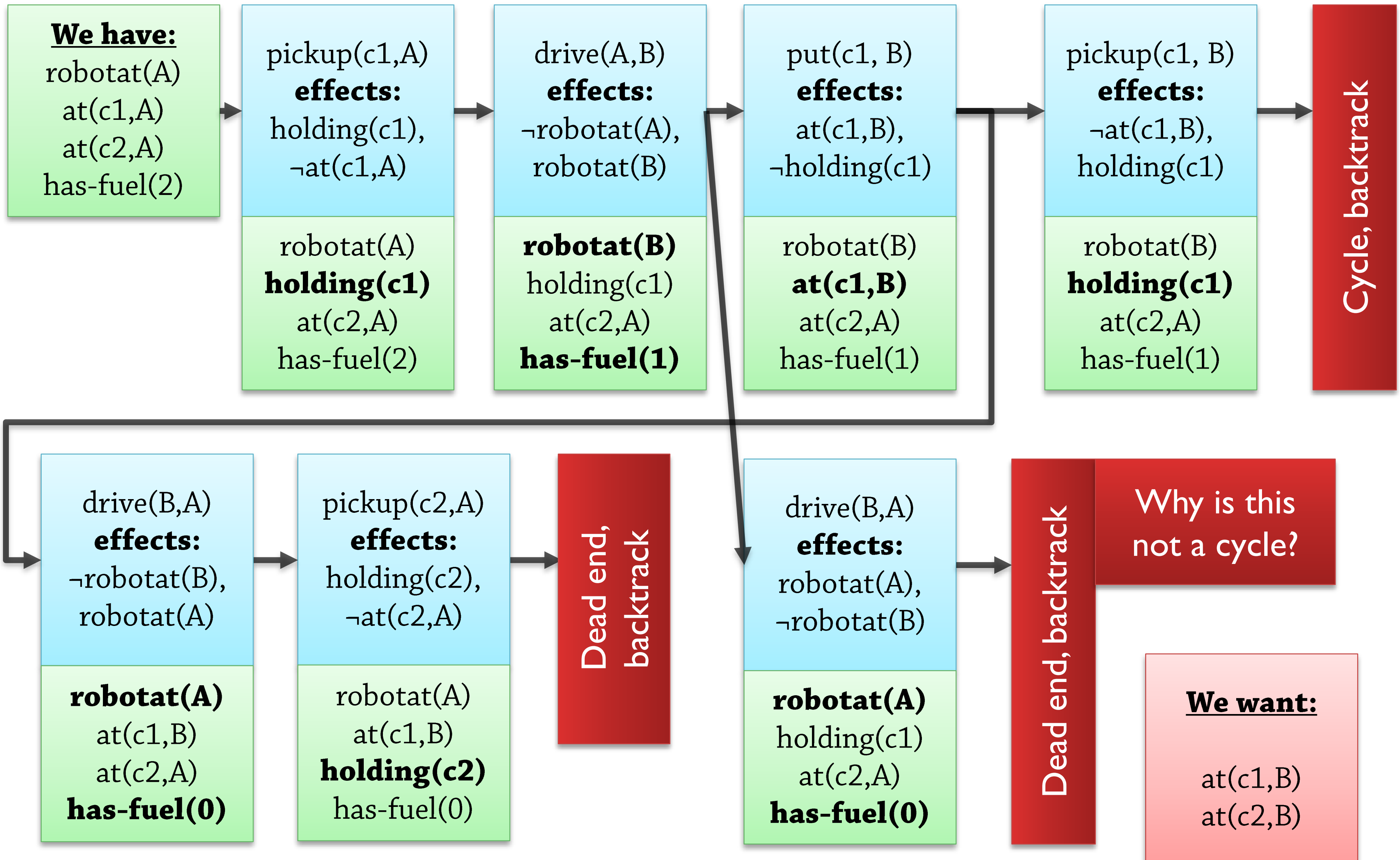
- Simple planning problem:
  - Two **<u>crates</u>**
    - At A
    - Should be at B



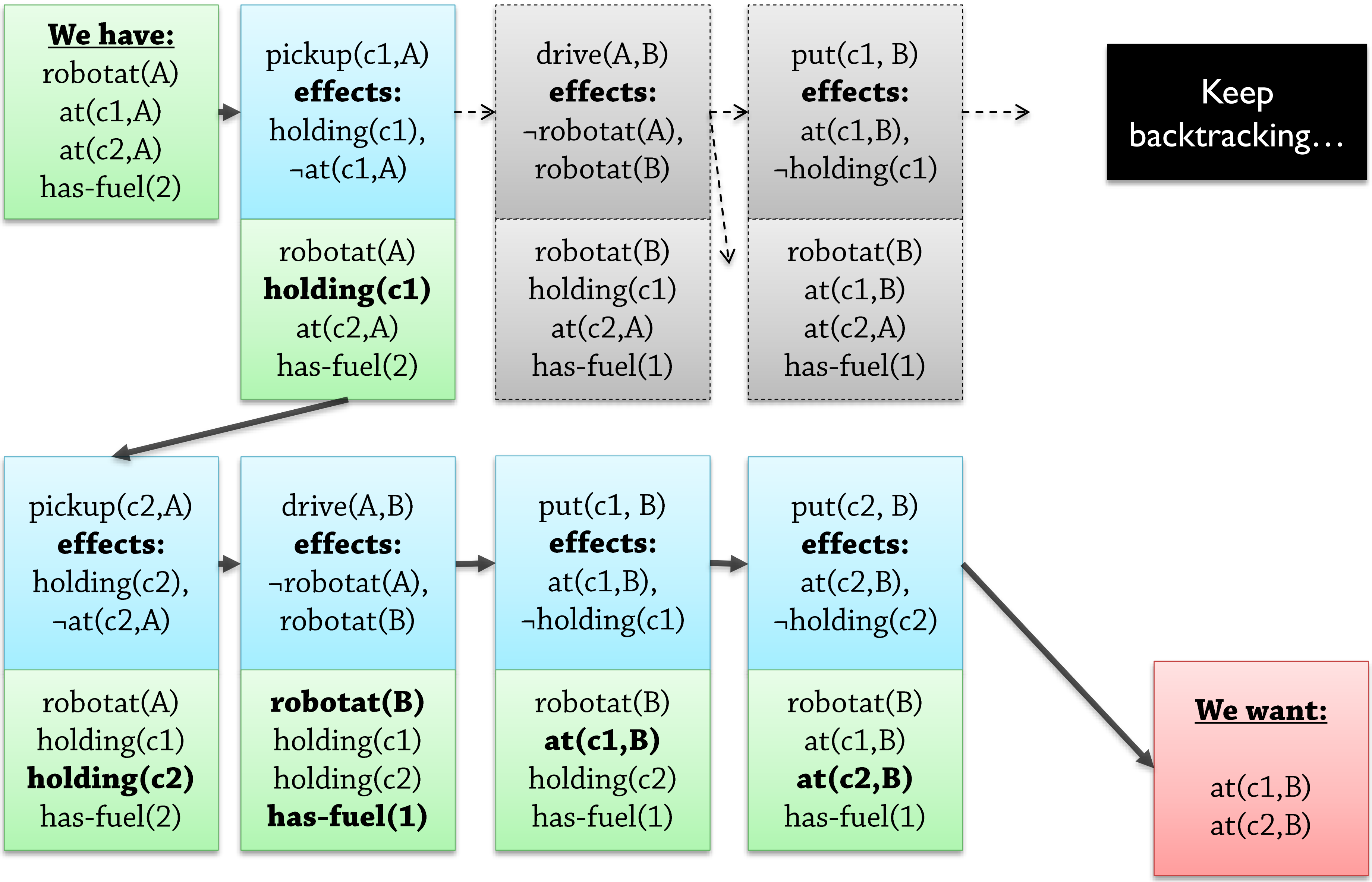
  - One **<u>robot</u>**
    - Can **<u>carry</u>** up to two crates
    - Can **<u>move</u>** between locations, which requires one unit of **<u>fuel</u>**
    - Has only two units of fuel

Let's see what a forward-chaining planner *might* do (depending on heuristics)…

Motivating Problem 2: Forward Search



Motivating Problem 3



Motivating Problem 4

- Observations:
  - **Most** actions we added before backtracking were **useful** and **necessary**!

    pickup(c1,A) | drive(A,B) | put(c1, B) | ~~drive(B,A)~~ | pickup(c2,A)

  - At first, we added them in the **wrong order**

    pickup(c1,A) | drive(A,B) | put(c1, B) | pickup(c2,A)

  - **Forward** and **backward** planning **commits** immediately to action order
    - Puts each action in its **final place** in the plan
  - State space **heuristics** must be smart enough to tell us:
    - Which actions are **useful**
    - ***When*** *to add them to the plan*

What if we could rearrange actions?

Partial Order Causal Link: Plan Structure

First Step: Insertion

- **<u>Sequences</u>** with arbitrary insertion: Useful?
  - Add actions in sequence, as in state space planning…

    pickup(c1,A) | drive(A,B) | put(c1, B)

  - Realize you need another one…

    pickup(c2,A) ← How to decide which action to *insert*, if not at the end

  - Make a space…

    How to decide where to insert it?

    pickup(c1,A) | ↓ | drive(A,B) | put(c1, B)

  - …and place the action there

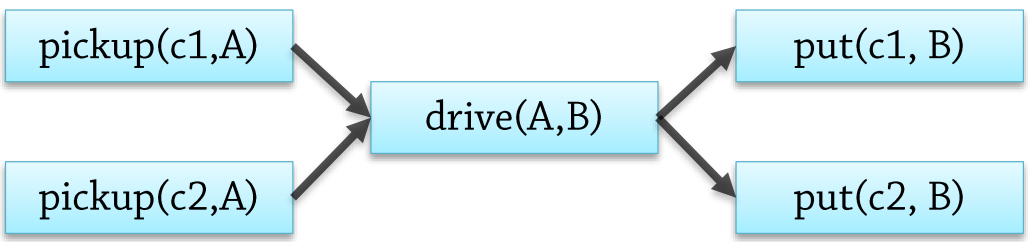
    pickup(c1,A) | pickup(c2,A) | drive(A,B) | put(c1, B)

    How to check that "**<u>old</u>**" preconditions remain satisfied?

Second Step: Partial Order

- If we must deal with this complexity:
  - We can "get more for the same price"

- Let's skip sequences completely – a plan could be *partially ordered*:

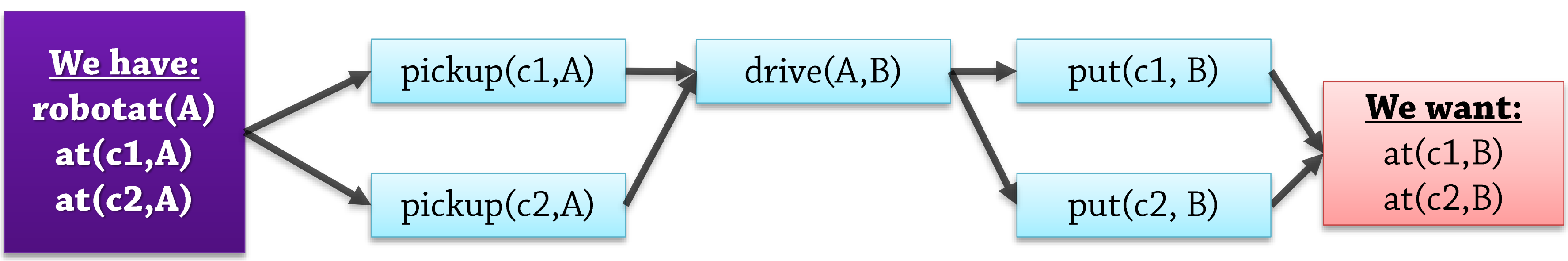


  - A set of **actions** $A = \{ a_1, a_2, a_3, \ldots \}$
  - A set of **precedence constraints** $\{ a_1 < a_2, \quad a_1 < a_3, \quad \ldots \}$
    - a_1 must finish before a_2 starts, …
    - Here: **solid arrows**

How do we *generate* these plans?

POCL 1: Introduction

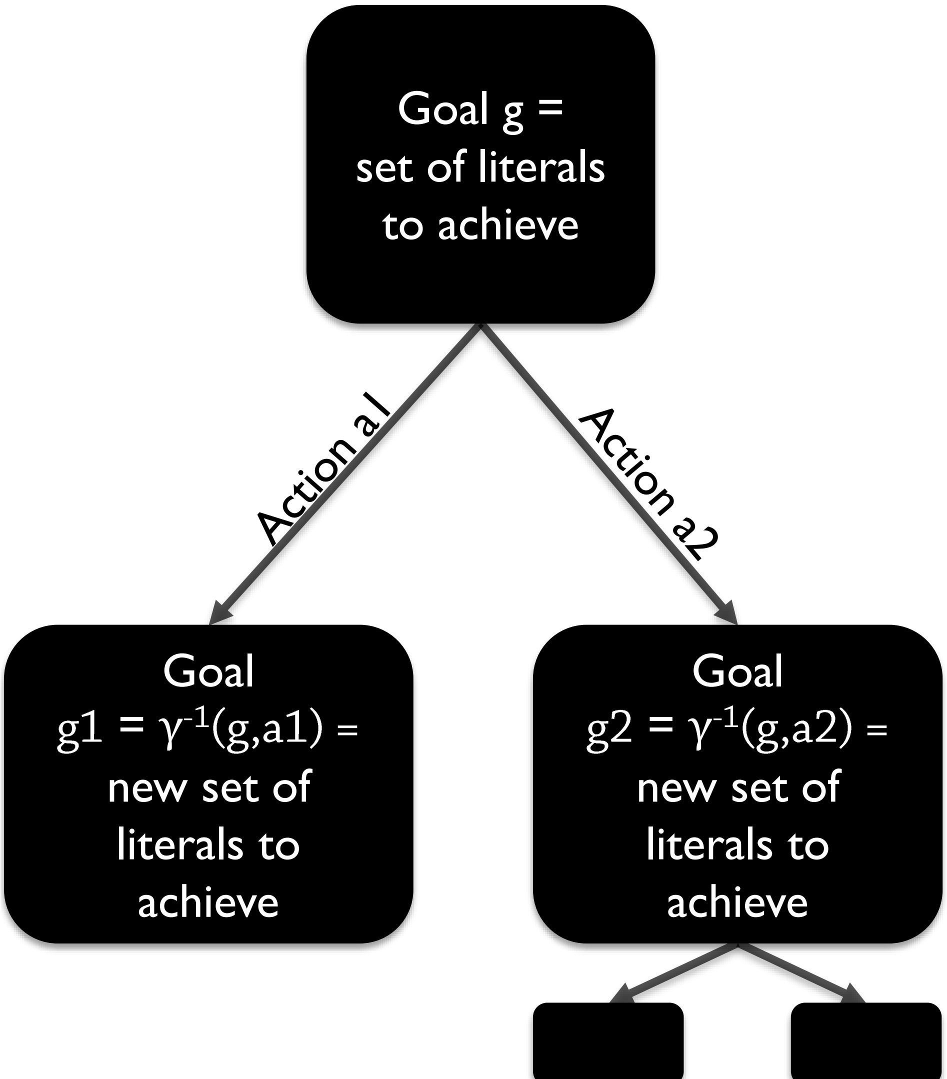
- **<u>Partial Order Causal Link</u>** (POCL) planning
  - Use a partial order, as described
    - Not when executing the plan
    - Only to **<u>delay commitment</u>** to ordering
  - As in backward search:
    - Add **<u>useful</u>** actions to achieve necessary conditions
    - Keep track of what **<u>remains</u>** to be achieved
    - But: Insert actions "**<u>at any point</u>**" in a plan



More sophisticated "bookkeeping" required!

POCL 2: Comparison to Backward Search

- Search tree for backward search, earlier:



The goal is a set of literals – simple!

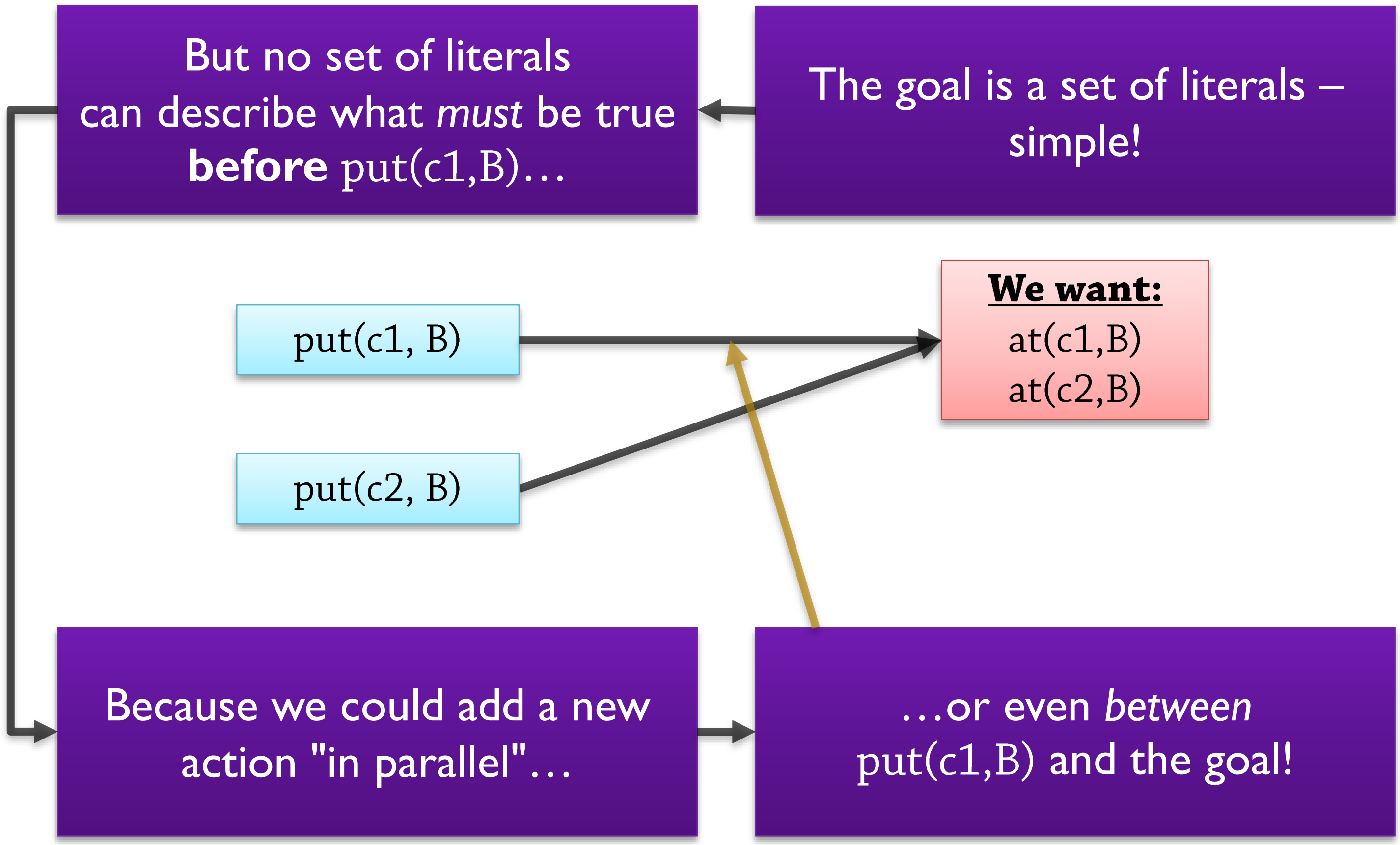
Every step takes you to a new set of literals to achieve

From a search node, you know how to reach the goal using a *sequence* of actions

A search node [2] can simply be a goal set

POCL 3: Comparison to Backward Search

- In POCL planning:
  - There is no sequence – and no clear "before" relation!

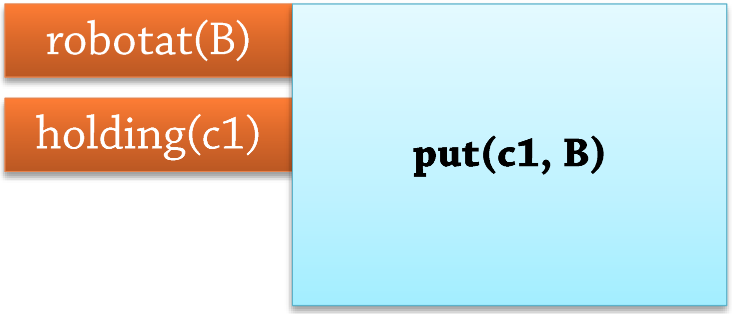


Has consequences for the POCL plan structure and the node structure…

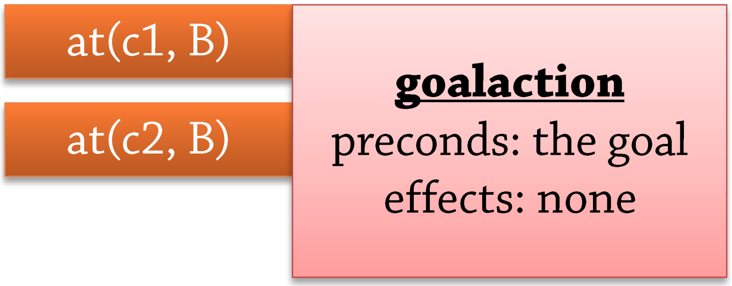
POCL 4: Conditions; Goal Action

- Must keep track of **individual propositions to be achieved**
  - Throughout the plan – not a single state g1 = γ^{-1}(g,a1)
  - May come from **preconditions** of every action in the plan

Notation chosen for this presentation: Preconditions on the left/top side



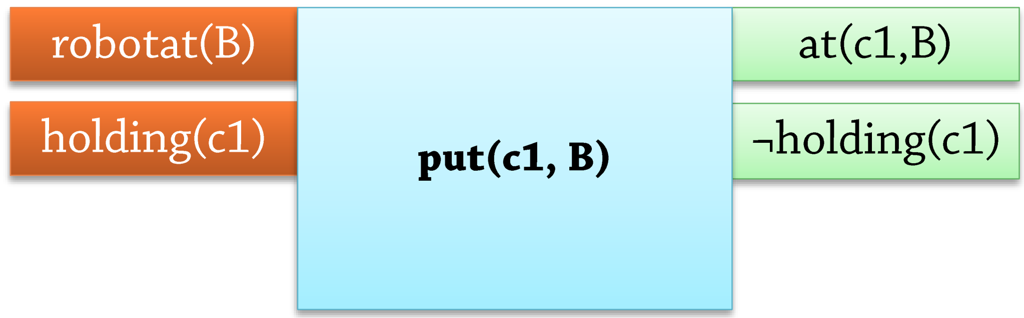
  - May come from the **problem goal** as in backward search
    - Trick: Use a **uniform representation**
    - Add a "fake" **goal action** to every plan, with the goals as preconditions!



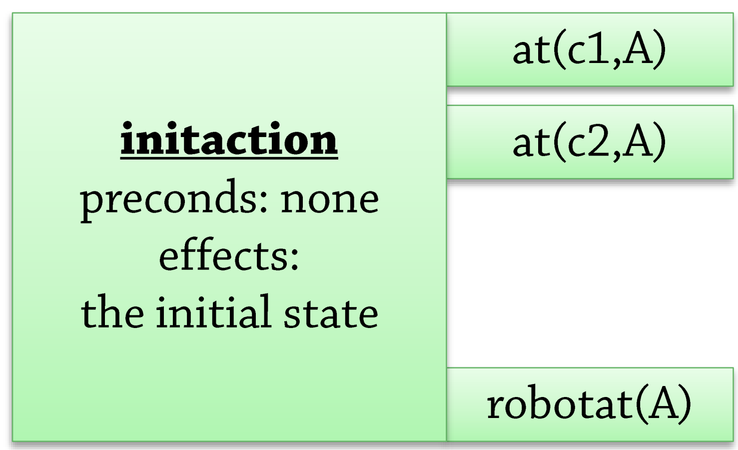
POCL 5: Effects; Initial Action

- Must keep track of **<u>individual propositions that are achieved</u>**
  - Throughout the plan – not from a single *relevant* action
  - May come from **<u>effects</u>** of every action in the plan

Notation chosen for this presentation: Effects on the right/bottom side



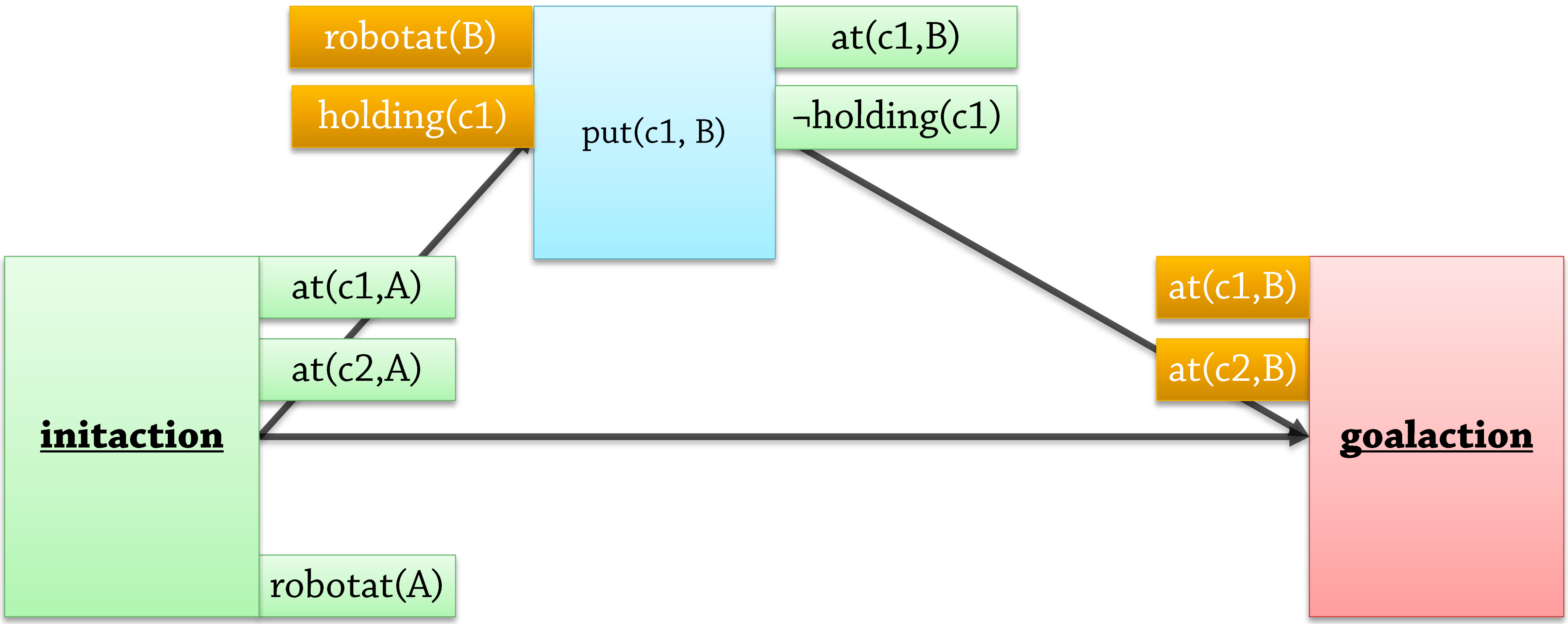
  - May come from the **<u>initial state</u>**
    - Trick: Add a "fake" **<u>initial action</u>** with the initial state as effects!



  - Effects are sometimes omitted from the slides, due to lack of space…

POCL 6: Precedence Constraints

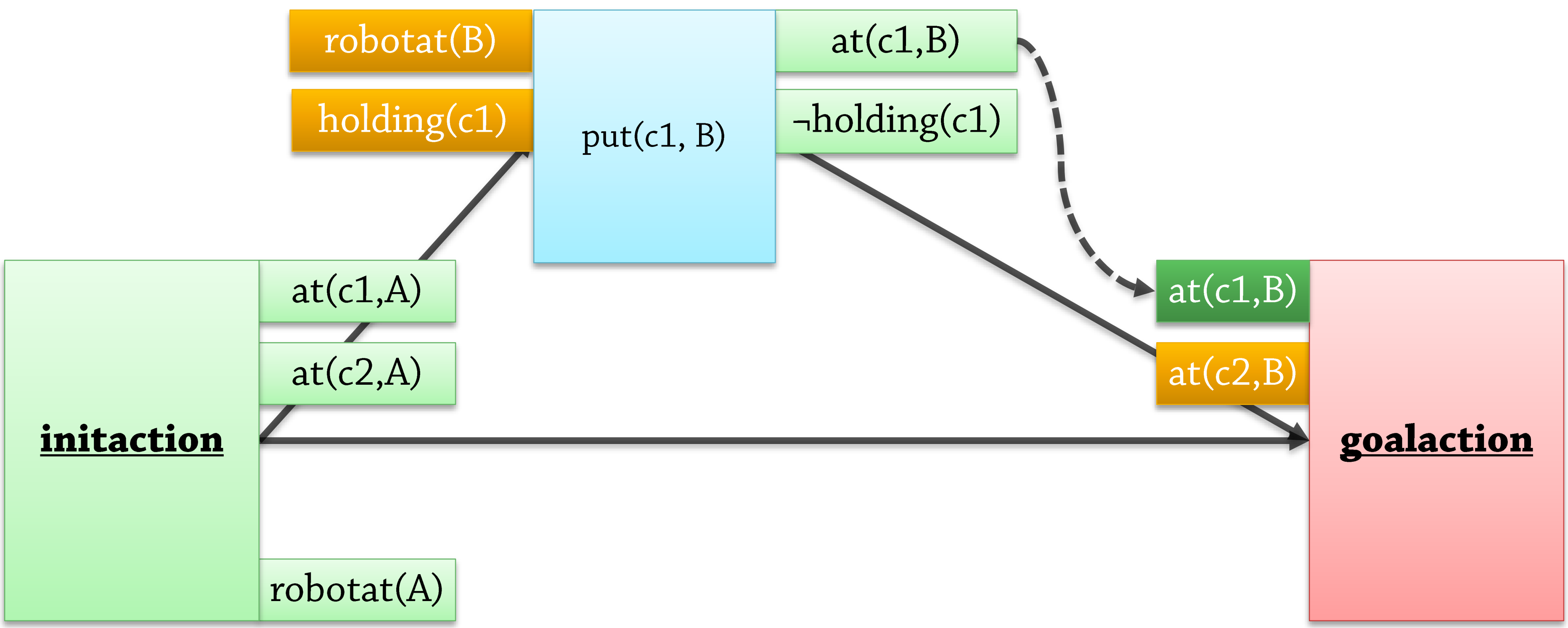
- Plan structure so far:



POCL 7: Causal Links

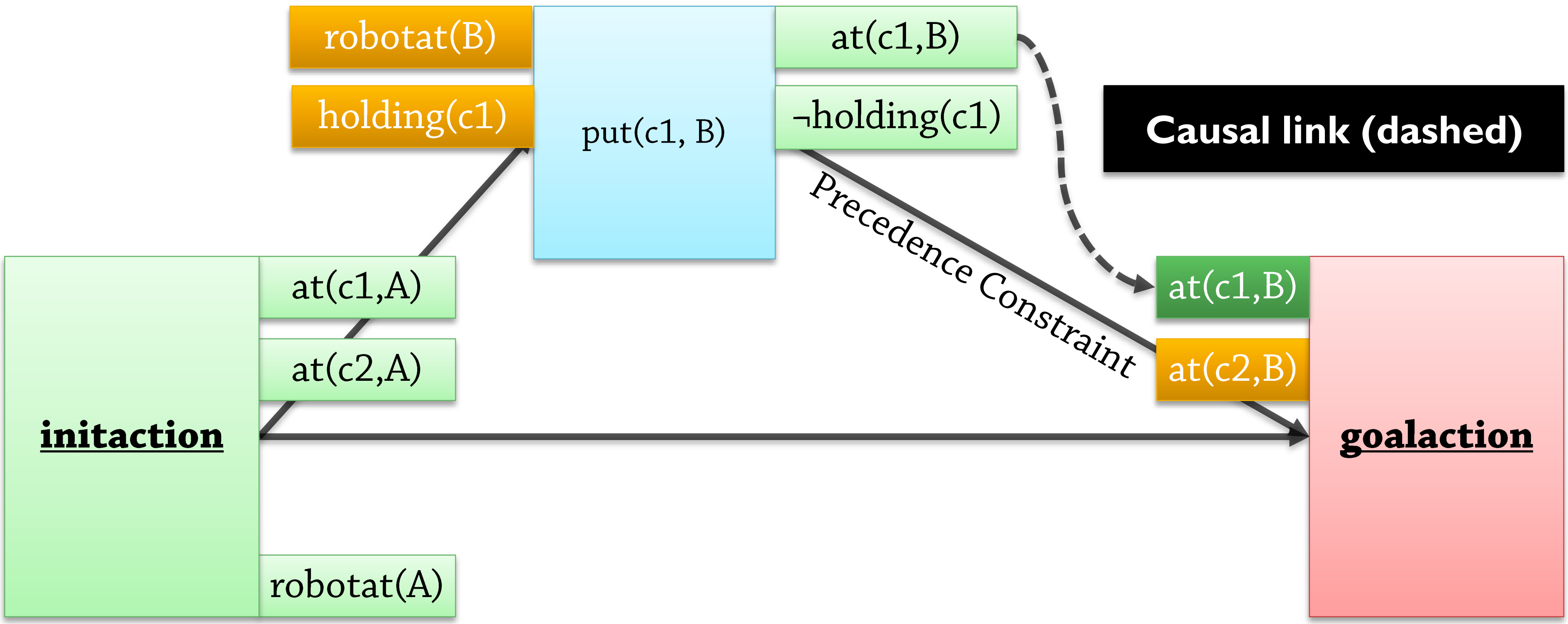
- Let's keep track of **which** action achieves **which** precondition
  - **Causal links**

Causal link (dashed):
at(c1,B) must **remain true**
between the end of *put* and the beginning of *goalaction*.
No one must delete it!
Important for *threat management* (later)



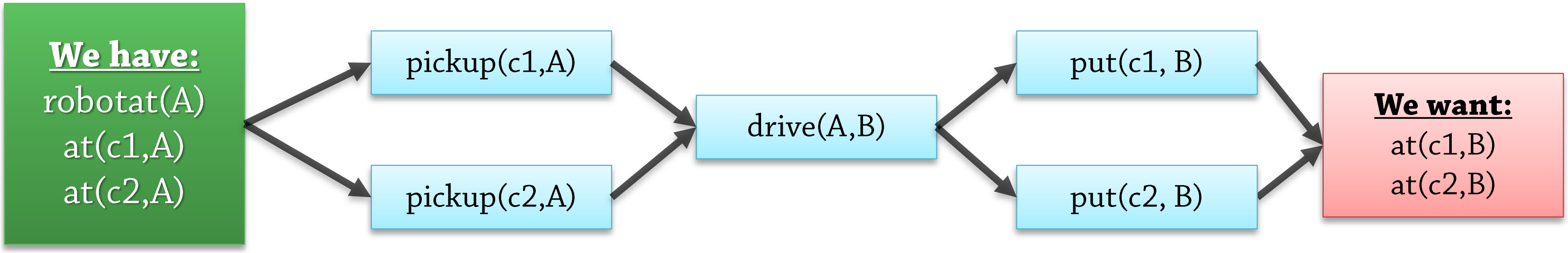
POCL 8: Partial-Order Plans

- To summarize, a ground **partial-order plan** consists of:
  - A set of **actions**
  - A set of **precedence constraints** $a \rightarrow b$
    - Action *a* must precede *b*
  - A set of **causal links** $a \xrightarrow{p} b$
    - Action *a* establishes the precond *p* needed by *b*



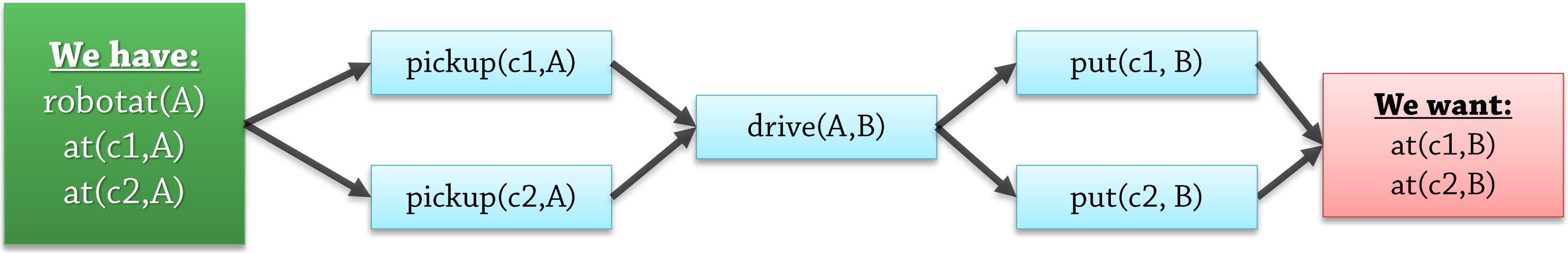
Partial-Order Solutions

- Original motivation: **performance**
  - Therefore, a partial-order plan is a **solution** iff **all** **sequential** plans satisfying the ordering are solutions
    - Similarly, **executable** iff corresponding sequential plans are executable
    - <pickup(c1,A), pickup(c2,A), drive(A,B), put(c1,B), put(c2,B)>
    - <pickup(c2,A), pickup(c1,A), drive(A,B), put(c1,B), put(c2,B)>
    - <pickup(c1,A), pickup(c2,A), drive(A,B), put(c2,B), put(c1,B)>
    - <pickup(c2,A), pickup(c1,A), drive(A,B), put(c2,B), put(c1,B)>



Partial Orders and Concurrency

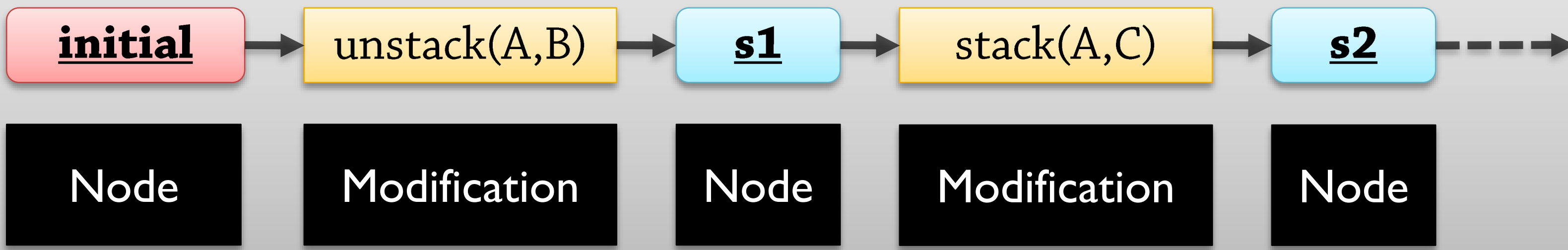
- Can be **extended** to allow **concurrent execution**
  - Requires a new formal model!
    - Our state transition model *does not define* what happens if c1 and c2 are picked up simultaneously!



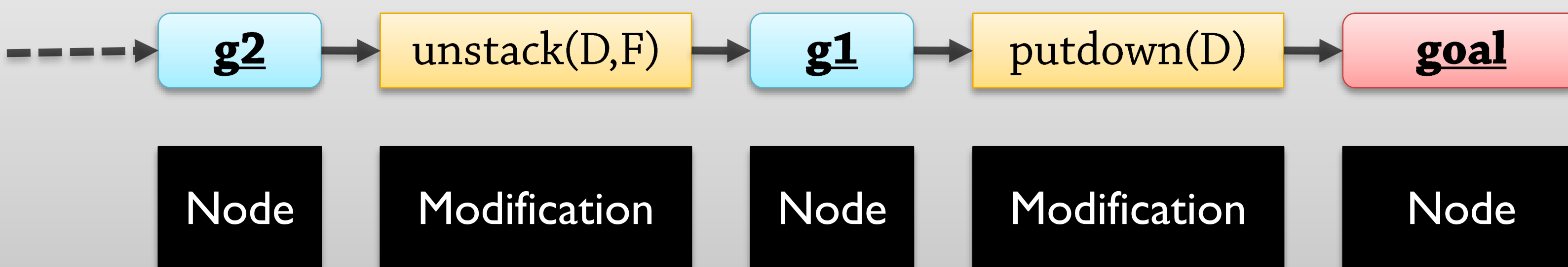
Partial Order Causal Link: Search Space

Context: Forward, Backward

Forward search: A search node is a "current state"

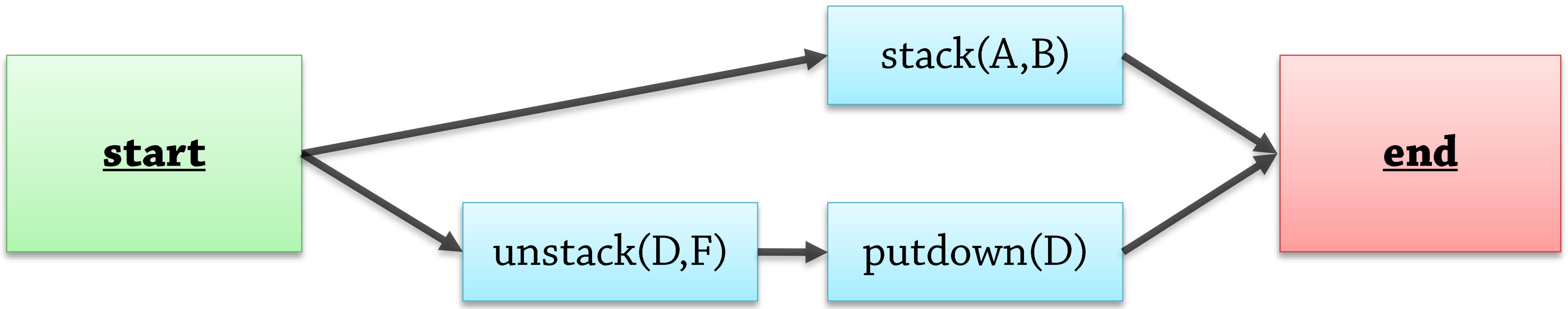


Backward search: A search node is a "current goal"



No Current State during Search!

- With **<u>partial-order plans</u>**: No "current" state or goal!
  - What is true after stack(A,B) below?
    - **<u>Depends</u>** on the order in which **<u>other</u>** actions are executed
    - **<u>Changes</u>** if we insert **<u>new</u>** actions **<u>before</u>** stack(A,B)!



A search node can't correspond to a state or goal!

Search Nodes are Partial Plans

- [1] Each node must contain *more information*: **The entire plan**!
  - [2] The **initial** search node contains an **initial plan**
    - The special **initial** and **goal actions**
    - A single **precedence constraint**

So: this is one form of ”plan-space” planning!

Initial node containing initial plan

Branching Rule

- **[3]** We need a **branching rule** as well!
  - Forward planning: One successor per action **applicable** in *s*
  - Backward planning: One successor per action **relevant** to *g*
  - POCL planning: ???

Could allow inserting <u>any</u> actions,
<u>any</u> precedence constraints…

Too much freedom, too many alternatives!

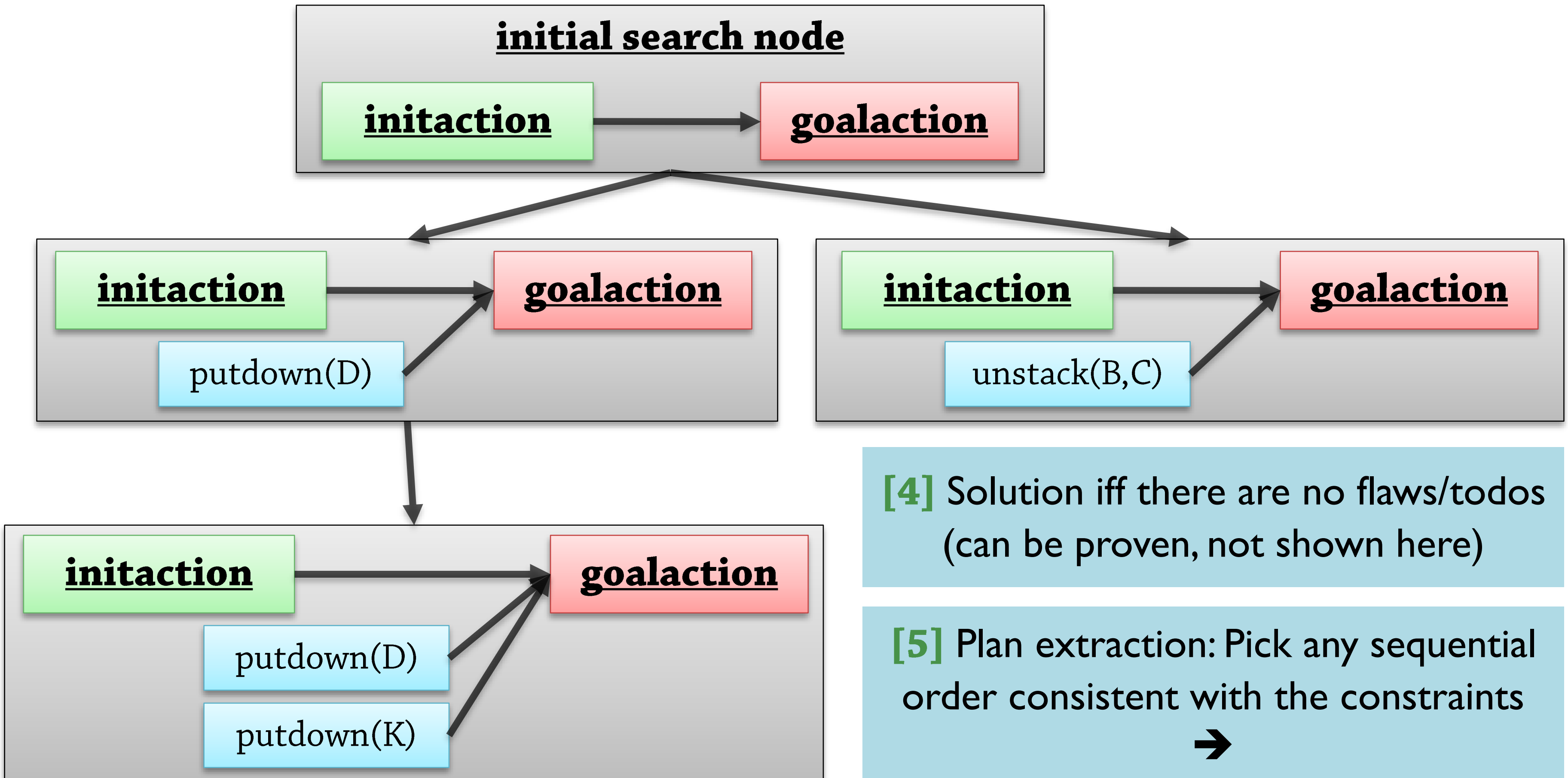
What do we <u>need</u> for completeness?

Branching Rule

- [3] **Branching rule** for POCL planning
  - Identify specific **reasons for modifying the plan**, called **flaws** (basically **todo:s)**
    - 1) **Open goal**: We haven't decided *how* to achieve a precondition – clear(A)
    - 2) **Threat**: An action may *interfere* with another
  - One successor for each different way of **repairing** a **flaw**

Search Space

- Gives rise to a **search space**
  - [6] Use search strategies, backtracking, heuristics, … to search **this** space!



[4] Solution iff there are no flaws/todos (can be proven, not shown here)

[5] Plan extraction: Pick any sequential order consistent with the constraints
➔
<putdown(D), putdown(K)> or
<putdown(K), putdown(D)>

Successors Repair Flaws: Open Goals and Threats

Flaws

- Flaw, *noun*:
  1. a feature that mars the perfection of something; defect; fault: beauty without flaw; the flaws in our plan.
  2. a defect impairing legal soundness or validity.
  3. a crack, break, breach, or rent.

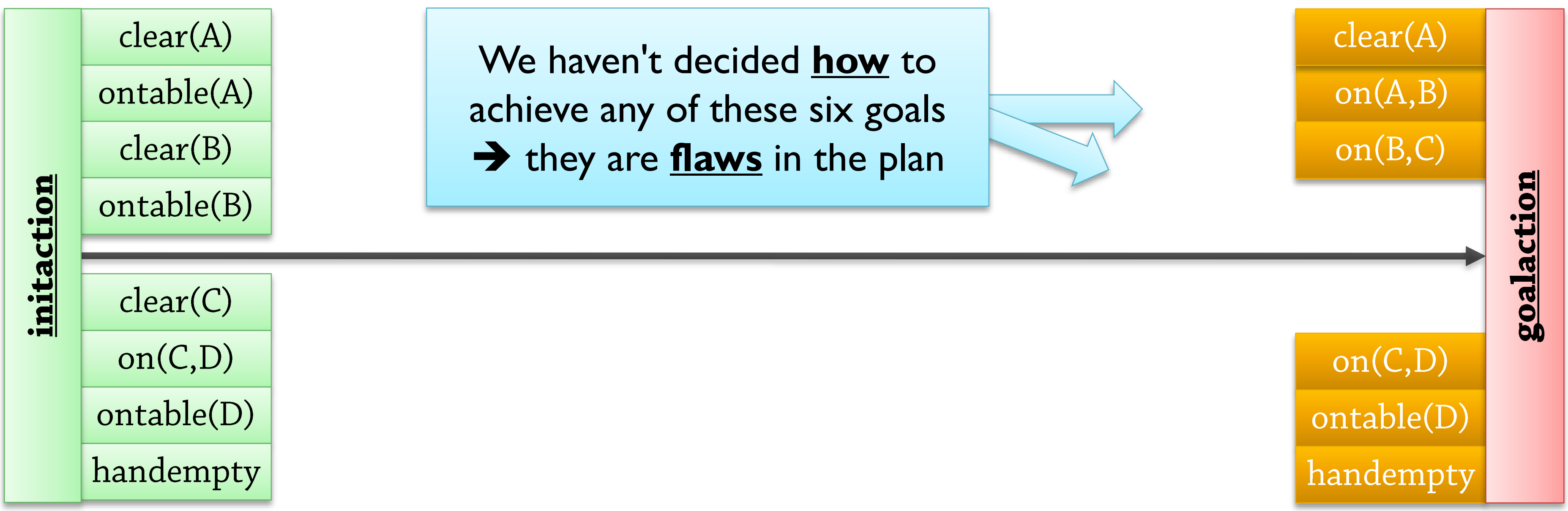
- Flaw, in POCL planning:
  - Something we **need to take care of** to **complete the plan**
  - Technical definition: An *open goal* or a *threat*

- **Not**:
  - Something that has "gone wrong"
  - A problem during planning
  - A mistake in the final solution
  - …

Open Goals

Flaw Type 1: Open Goals

- Open goal:
  - An **action** *a* has a **precondition** *p* with **no incoming causal link**



clear(A) **is** already true in s0, but there is no causal link…

Adding one from s0 means clear(A) **must never be deleted**!
We need other alternatives too: Delete clear(A), then re-achieve it for goalaction…

Flaw Type 1: Open Goals

- To **resolve** an open goal :
  - Find an action *b* that causes *p*
    - Can be a *new* action
    - Can be an action *already* in the plan, if we can *make* it precede *a*
  - Add a **causal link**

Partial order! This was not possible in backward search…

Essential:
Even if there **is already** an action that causes p,
you can still add a **new** action that **also** causes p!

Resolving Open Goals 1

- Here: **<u>Six</u>** open goals
  - Could choose to **<u>find support for clear(A)</u>**:
    - From *initaction*
    - From a new unstack(B,A), unstack(C,A), or unstack(D,A)
    - From a new stack(A,B), stack(A,C), stack(A,D), or putdown(A)
  - Could choose to **<u>find support for on(A,B)</u>**:
    - Only from a new instance of stack(A,B)
  - …

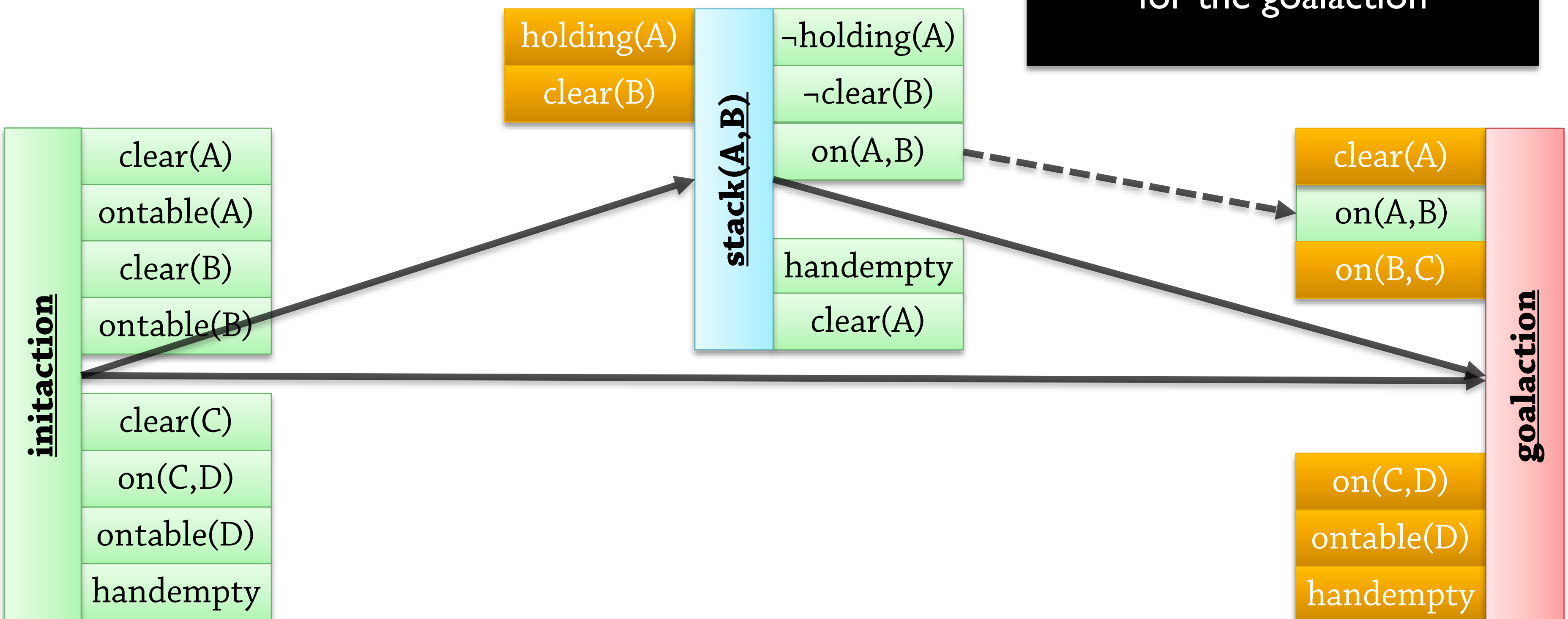
8 distinct successors

+1 successor

Resolving Open Goals 2

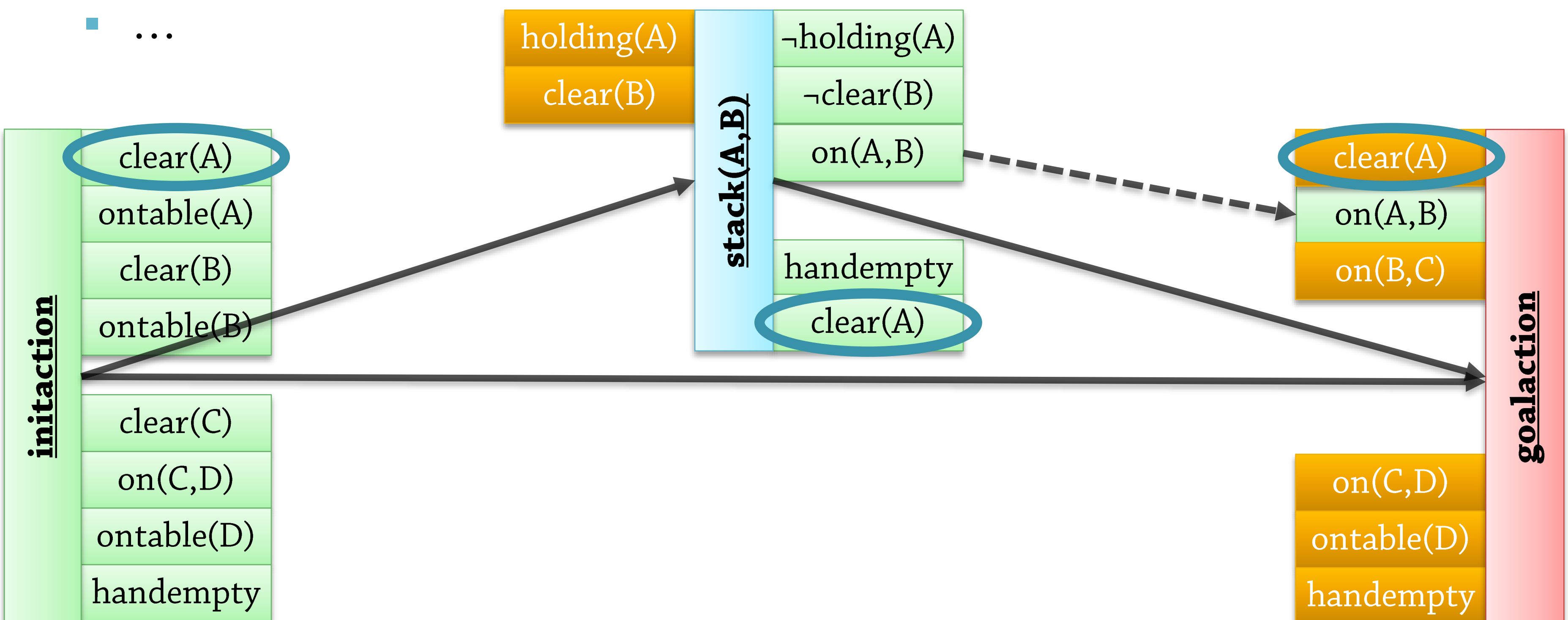
- Suppose we **add stack(A,B)** to **support (achieve) on(A,B)**
  - Must add a **causal link** for **on(A,B)**
    - Dashed line
  - Must **also** add precedence constraints
  - *Looks* totally ordered
    - Because it actually only has one "real" action…

Causal link says:
This instance of stack(A,B)
is responsible for
achieving on(A,B)
for the goalaction



Resolving Open Goals 3

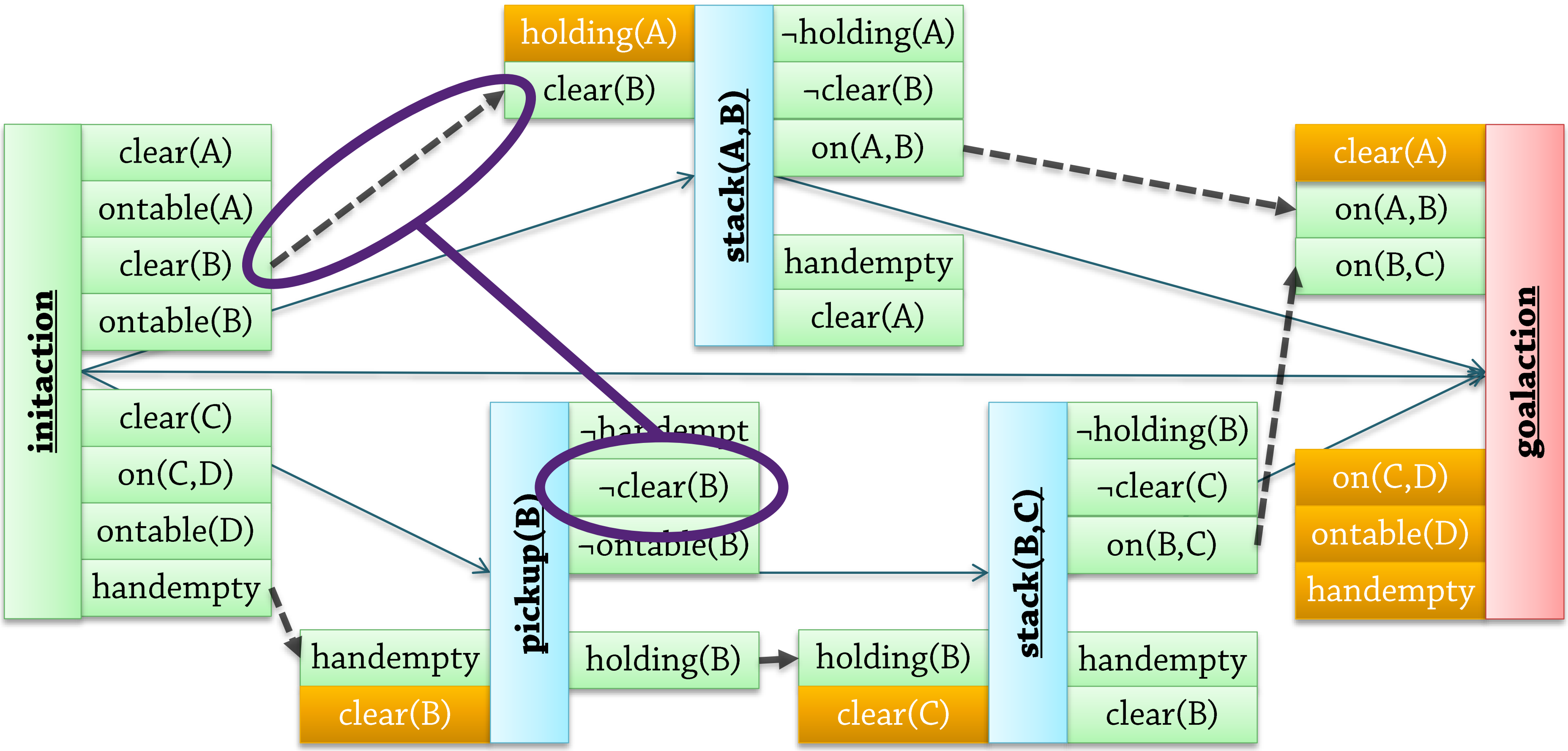
- Now: **7** open goals (one more!)
  - Can choose to find support for clear(A):
    - From the initaction
    - From the instance of **<u>stack(A,B)</u>** that we just added
    - From a **<u>new</u>** instance of **<u>stack(A,B)</u>**, stack(A,C), stack(A,D), or putdown(A)
    - From a **<u>new</u>** instance of unstack(B,A), unstack(C,A), or unstack(D,A)
  - …



Threats

Flaw Type 2: Threats

- Second flaw type: A **threat against a causal link**
  - initaction **should support** clear(B) for stack(A,B) – there's a **causal link**
  - pickup(B) **deletes** clear(B), and may occur **between** initaction and stack(A,B)
  - So we can't be certain that clear(B) still holds when stack(A,B) starts!



Flaw Type 2: Threats (2)

- Another way of **<u>illustrating</u>** a threat, on a "timeline"

→ time

Action A1 executes here, <u>makes</u> clear(B) true

Action A2 executes here, <u>requires</u> clear(B)

Action A3 unconstrained, may execute here: No problem

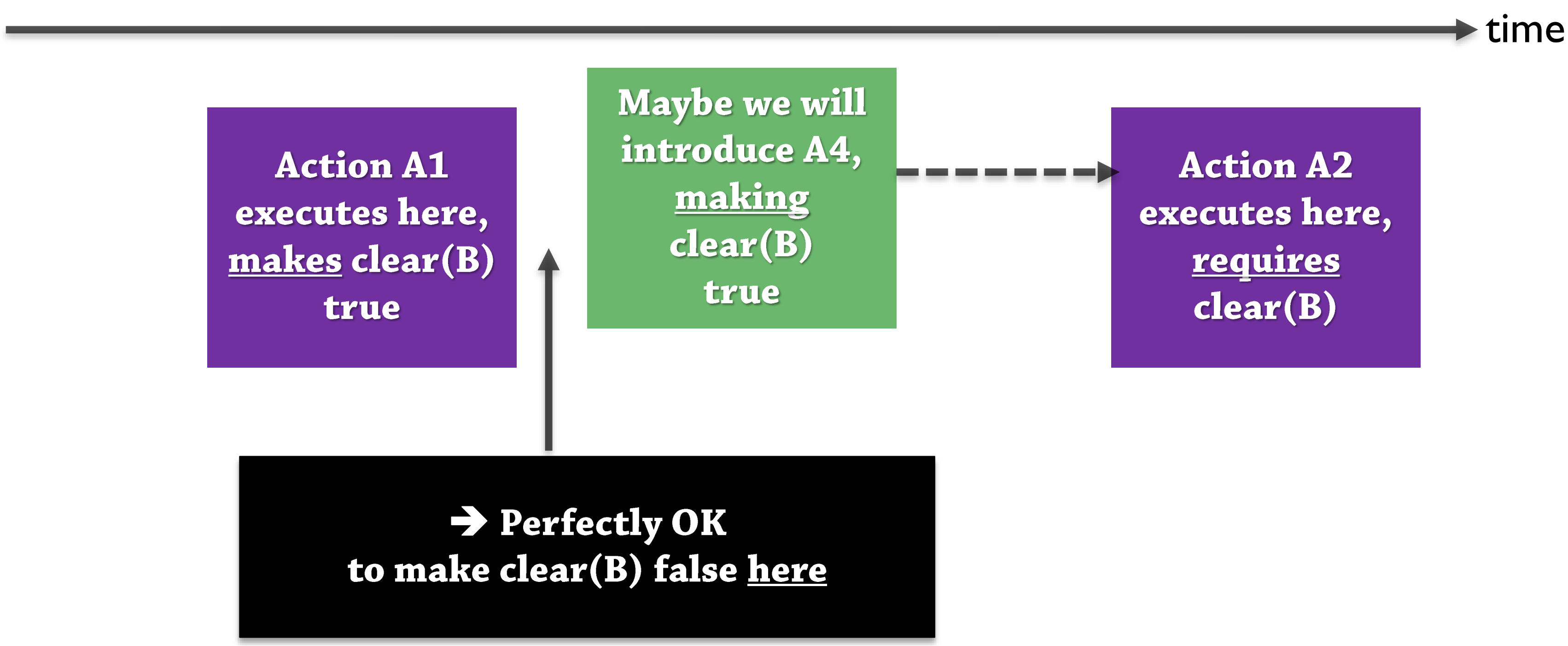
Action A3 unconstrained, may execute here → problem?

Action A3 unconstrained, may execute here: No problem

No! There is no causal link, so no reason to assume clear(B) must be preserved from A1 to A2!

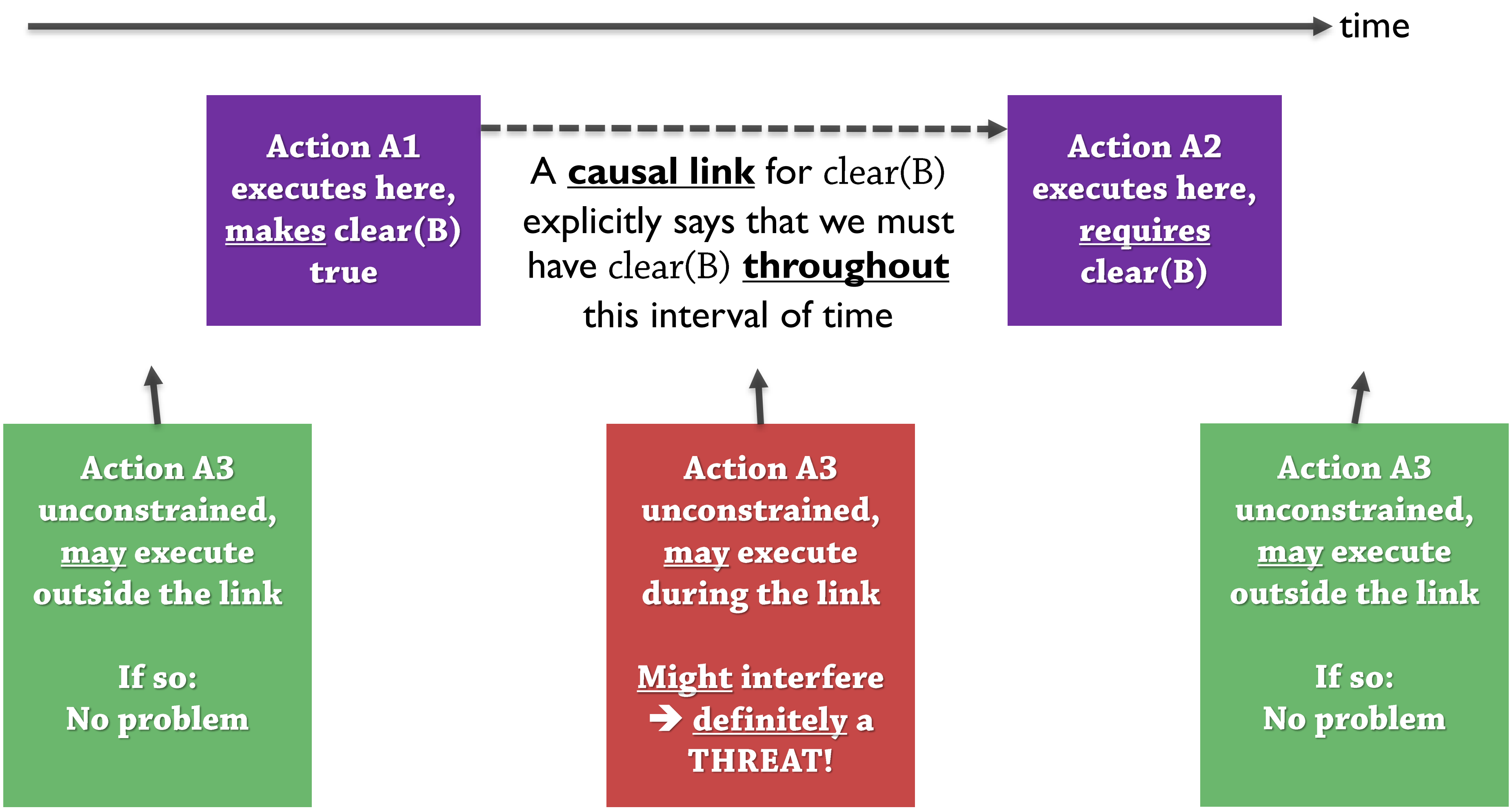
Flaw Type 2: Threats (3)

- Why no threat without causal link?



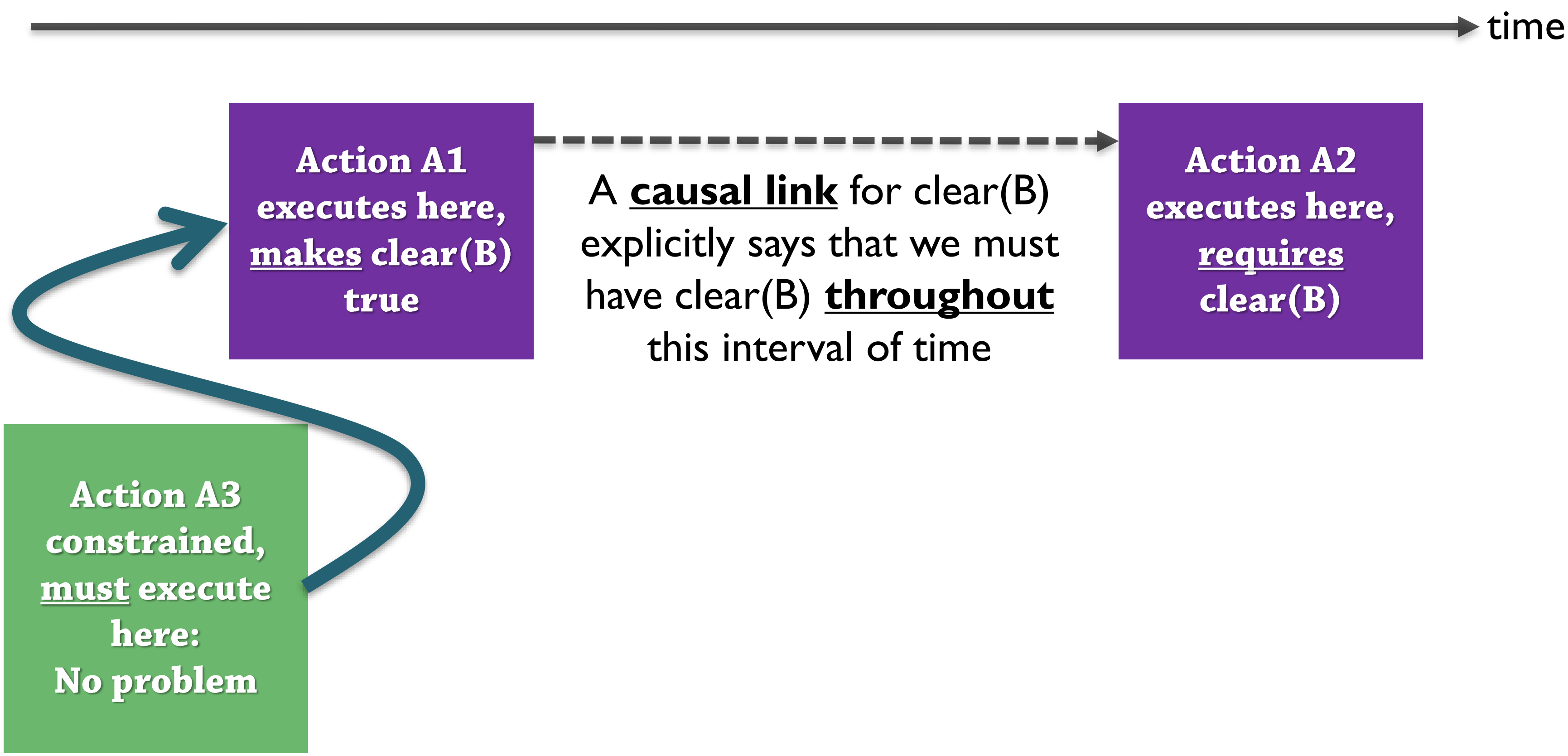
Flaw Type 2: Threats (4)

- But when we have a causal link:



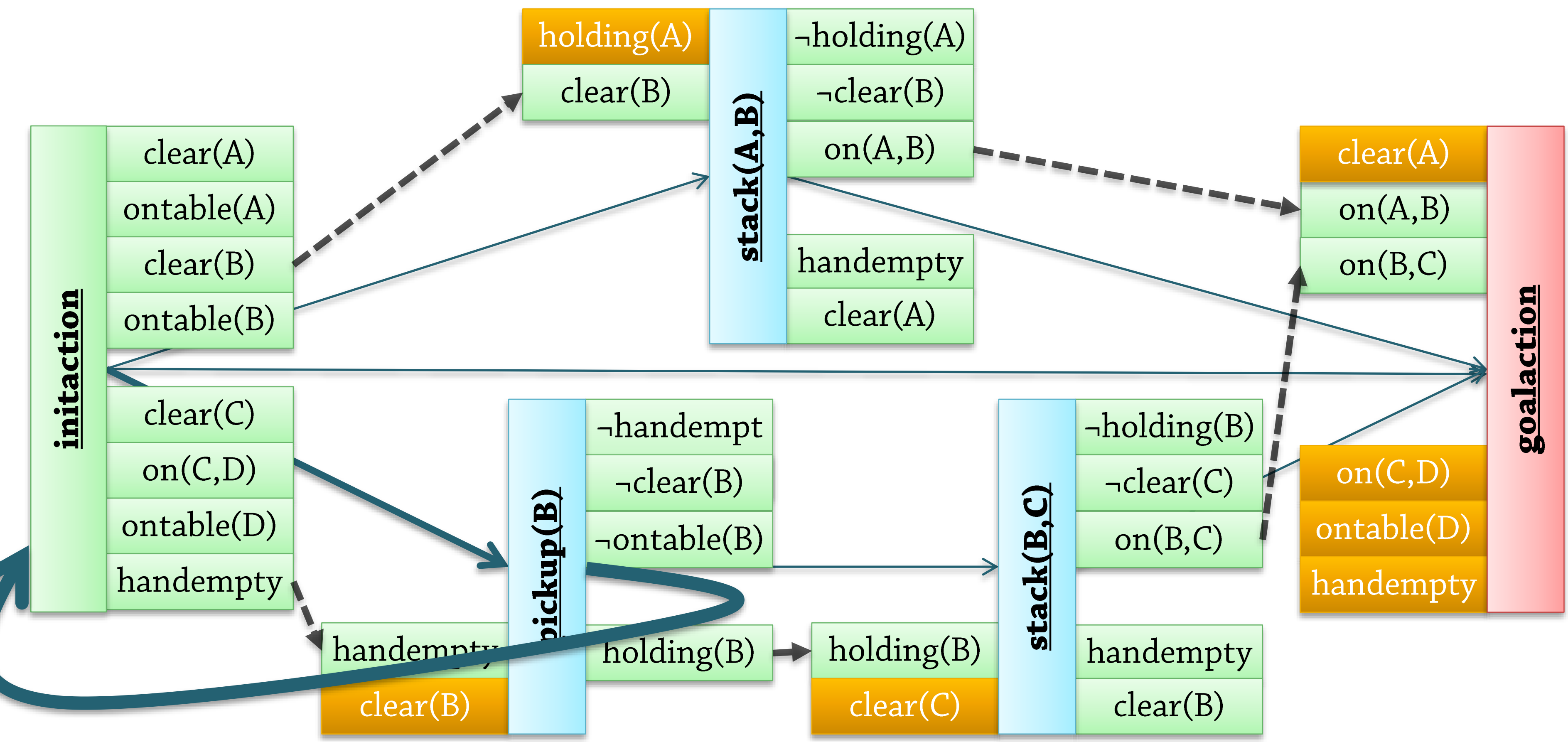
Resolving Threats (1)

- Resolution 1: The action that **disturbs** the causal link is placed **before** the action that **supports** the precondition
  - Only possible if the resulting partial order is consistent (acyclic)!



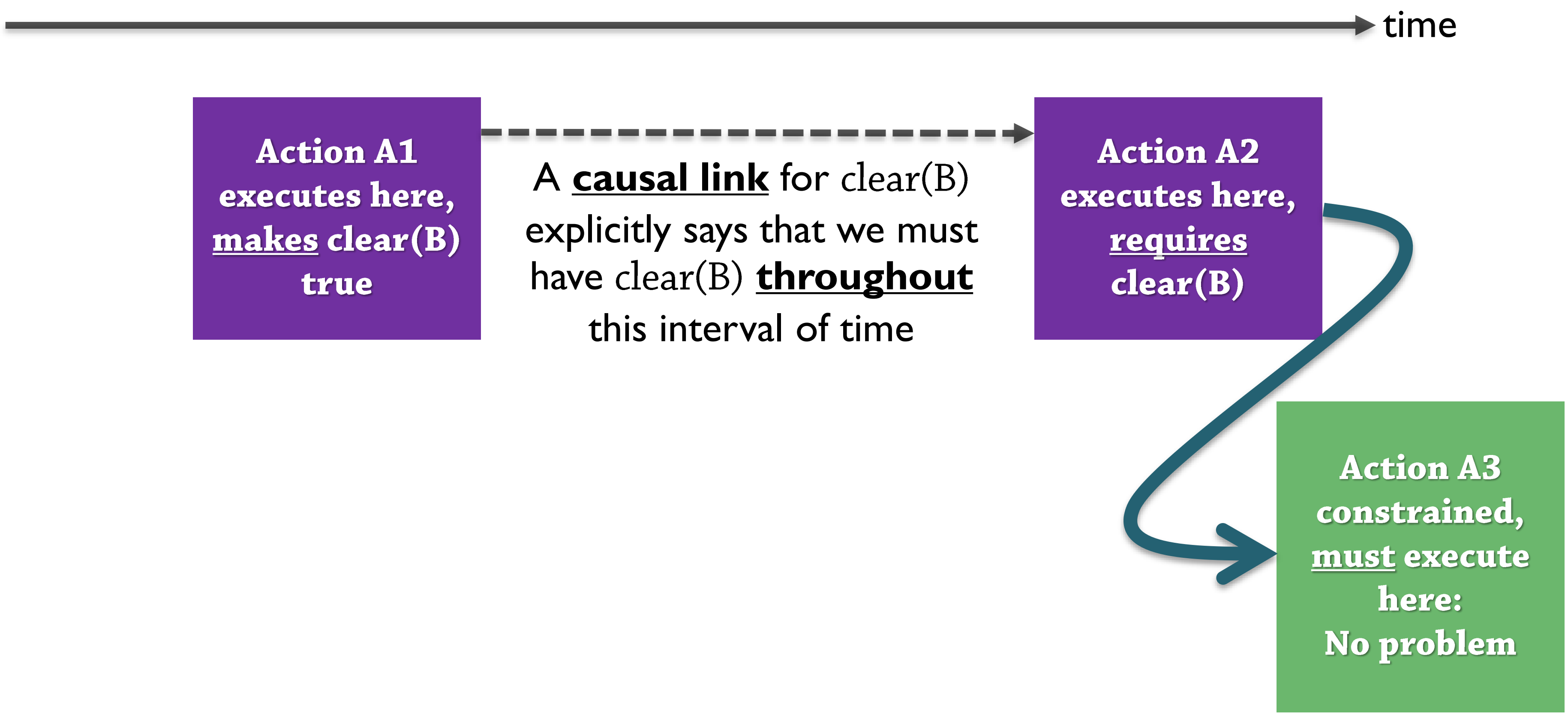
Resolving Threats (2)

- In this case, not consistent



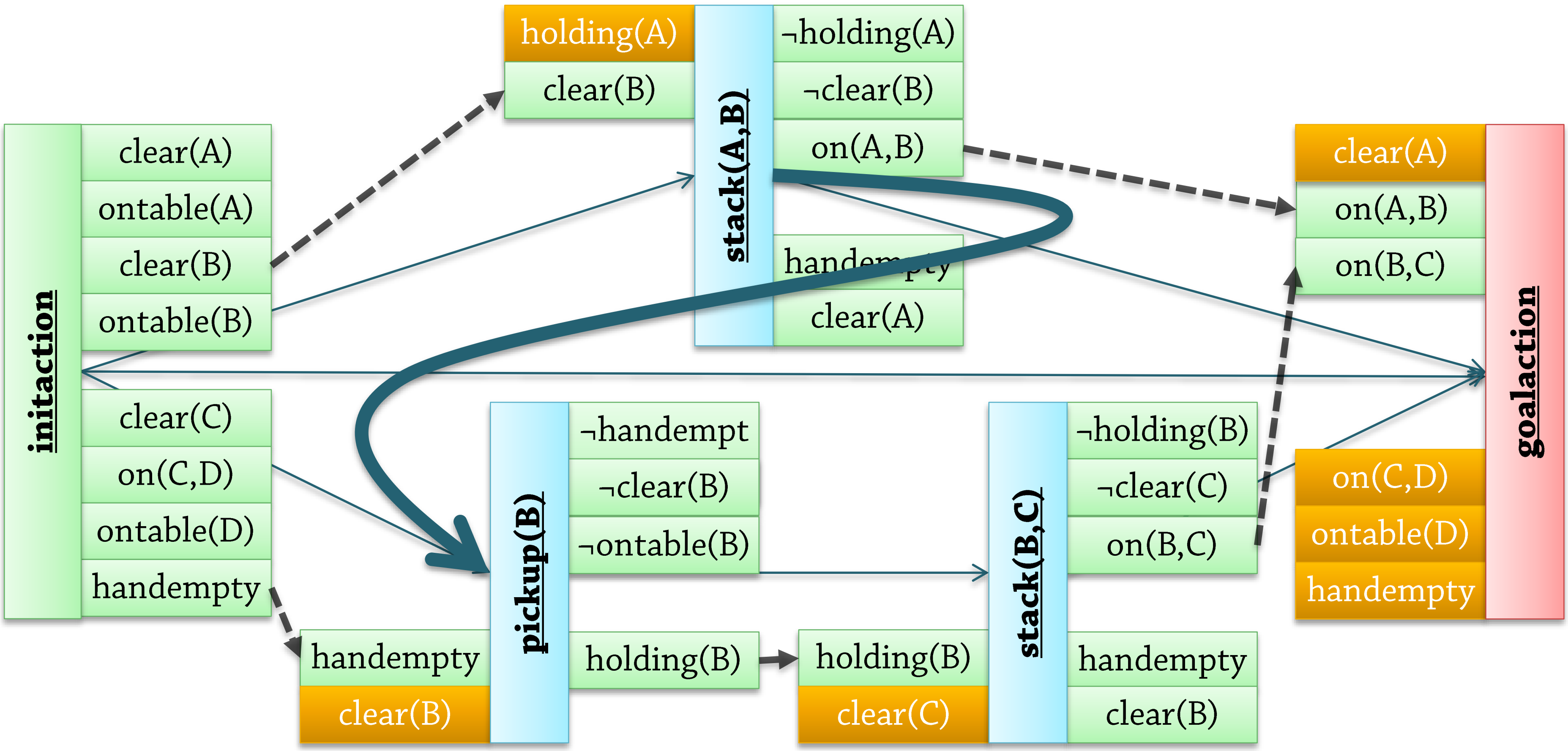
Resolving Threats (3)

- Resolution 2: The action that **disturbs** the causal link is placed **after** the action that **requires** the precondition
  - Only possible if the resulting partial order is consistent (acyclic)!



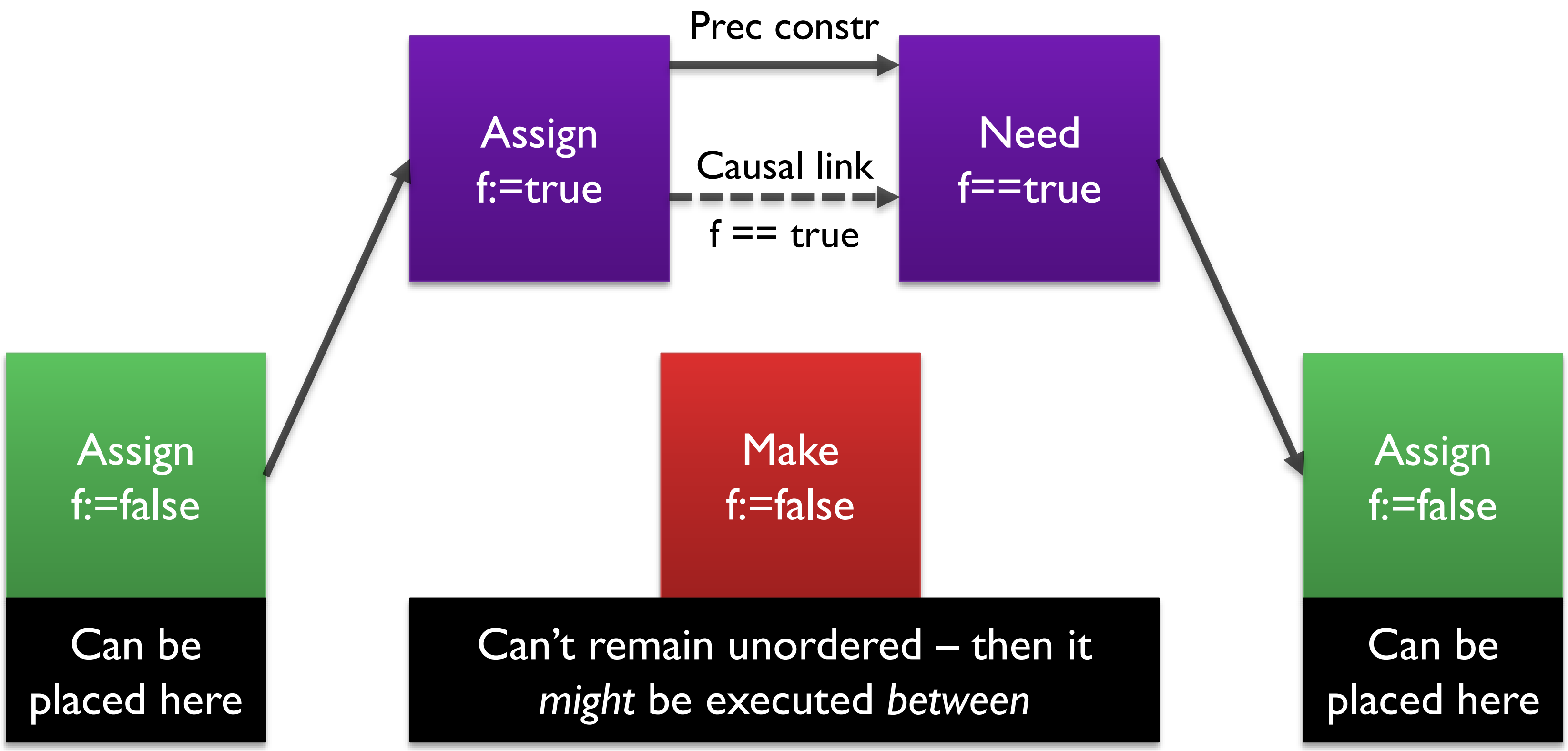
Resolving Threats (4)

- Works for this example



Resolving Threats (5)

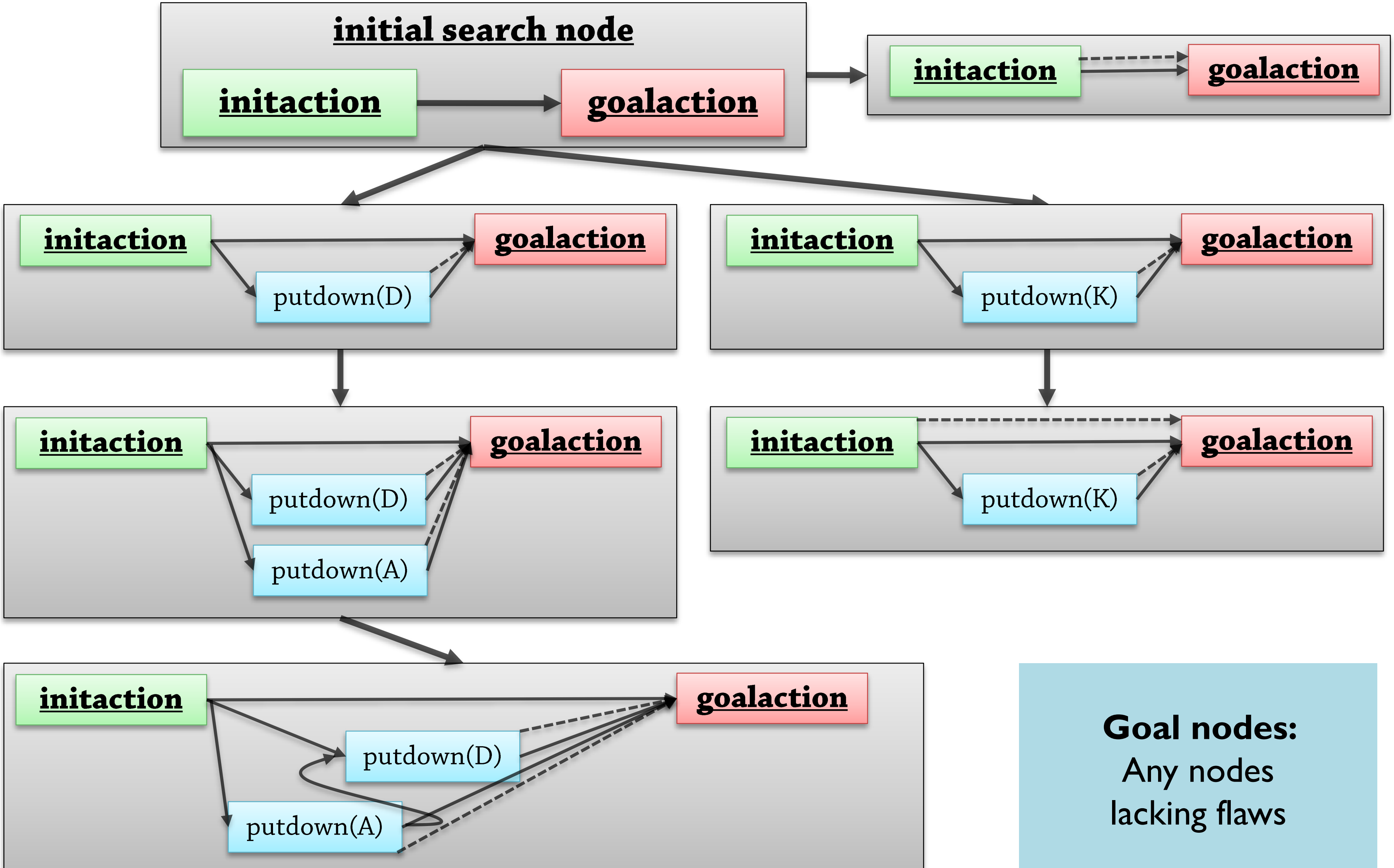
- Summary:



POCL Algorithms

Repetition: POCL Search Space

jonkv@ida



Goal nodes:
Any nodes
lacking flaws

Repetition: Planning as Search

- **Repetition**
  - **search**(*problem*) {

```
search(problem) {
    initial-node ← make-initial-node(problem) // [2]
    open ← { initial-node }
    while (open ≠ ∅) {
        node ← search-strategy-remove-from(open) // [6]
        if is-solution(node) then  // [4]
            return extract-plan-from(node) // [5]

        foreach newnode ∈ successors(node) { // [3]
            add newnode to open
        }
    }
    // Expanded the entire search space without finding a solution
    return failure
}
```

[3] Expand Successors: **All** ways of resolving **some** flaw

POCL planning

- **POCL planning – one possible formulation** (sound/complete):

```
pocl-search(problem) {
    initial-node ← make-initial-node(problem.initial, problem.goal) // [2]
    open ← { initial-node }
    while (open ≠ ∅) {
        π ← search-strategy-remove-from(open) // [6]
        flaws ← OpenGoals(π) ∪ Threats(π)
        if flaws = ∅ then
            return π // [5]

        select any flaw φ ∈ flaws
        resolvers ← FindResolvers(φ, π)  // May be the empty set!
        foreach r in resolvers {
            π' ← Refine(r, π) // Actually apply the resolver
            open ← open ∪ { π' }
        }
    }
    return failure
}
```

[4] Can prove: *π* is a solution iff there are no remaining flaws

One flaw chosen!

But **all** resolvers must be tested…

[3] Expand Successors: **All** ways of resolving **some** flaw

Returns a **partially** ordered solution plan
Any **total** ordering of this plan will achieve the goals

Understanding Successors in POCL

select any flaw φ ∈ *flaws*
resolvers ← FindResolvers(φ, π)

foreach r in *resolvers*:
  π' ← Refine(ρ, π)
  open ← *open* ∪ { π' }

At first we said *'every flaw leads to successors'.*

But it is actually **sufficient** to try **one** flaw, **any** flaw, to resolve.
Testing other flaws would be *redundant*. **Why?**

1) Every flaw *has* to be resolved
2) Choosing this flaw *later* cannot help us resolve it: All possibilities already exist
3) Choosing this flaw *later* cannot help us resolve some other flaw

Allows us to use heuristics to **select flaws**
(as well as **prioritizing open nodes**)

We **must** allow search to test different resolvers for the chosen flaw. **Why?**

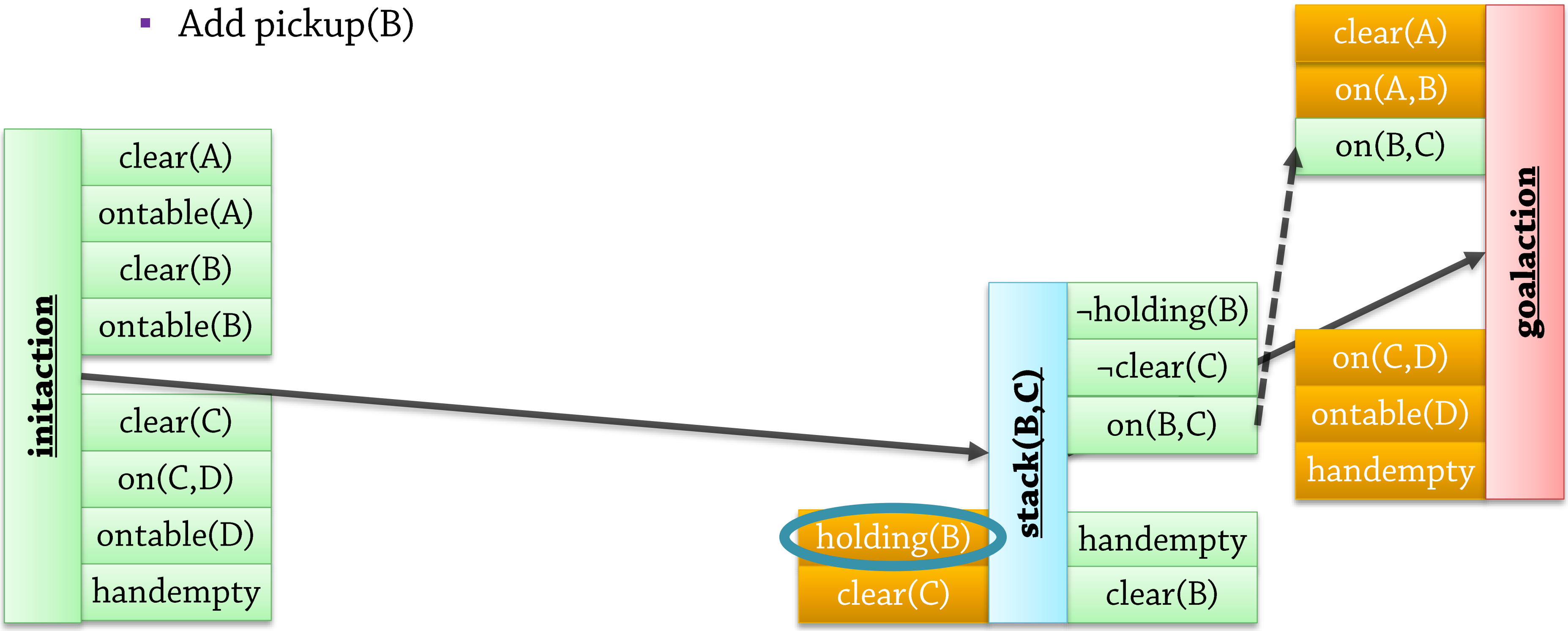
1) Choosing one resolver can prevent other problem resolutions.
2) Open goal: Use action A or B?
3) Threat: Which order to choose?

Lifted Planning Revisited

Partial Instantiation

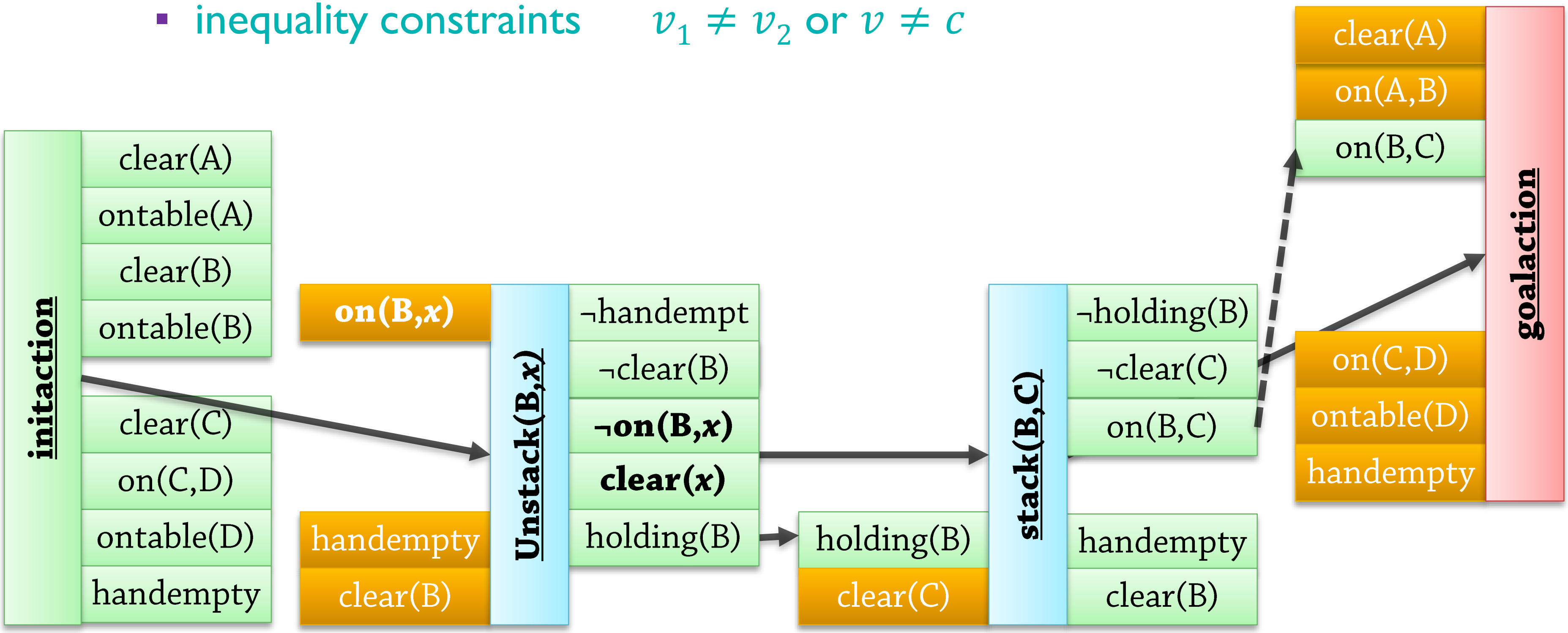
- Suppose we want to achieve holding(B)
  - **Ground** search generates **many** alternatives
    - Add unstack(B,A), unstack(B,F), unstack(B,G), …
    - Add pickup(B)
  - **Lifted** search generates two **partially instantiated** alternatives
    - Add **unstack(B, *x*)**
    - Add pickup(B)

So far, we see no reason why we should unstack B from any **specific** block!



Partial-Order Plans

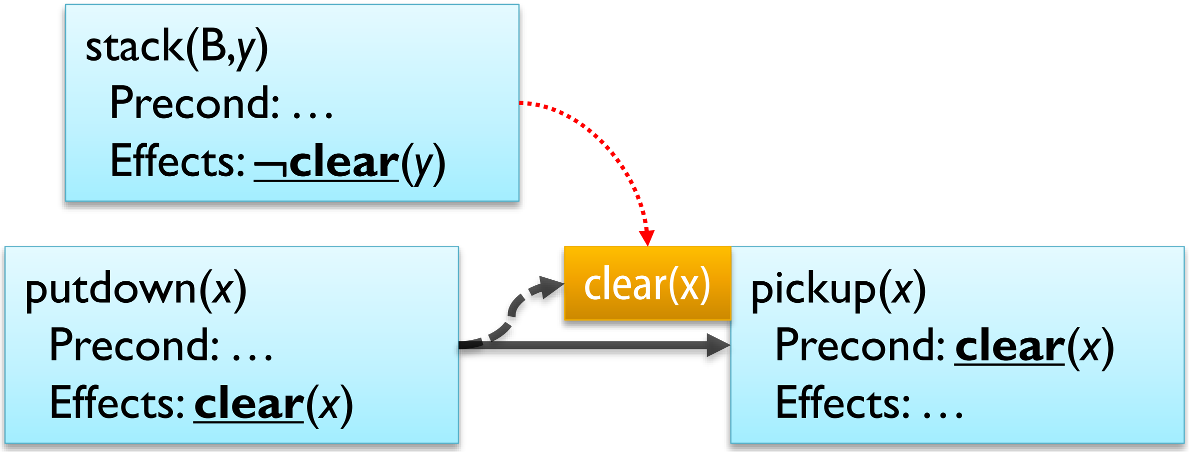
- A **lifted partial-order plan** consists of:
  - A set of **possibly unground actions**
  - A set of **precedence constraints**: *a* must precede *b*
  - A set of **causal links**: action *a* establishes the precond *p* needed by *b*
  - A set of **binding constraints**:
    - equality constraints $v_1 = v_2$ or $v = c$
    - inequality constraints $v_1 \neq v_2$ or $v \neq c$



Resolving Threats

jonkv@ida

- Another way of **resolving threats** for lifted plans:
  - For partly uninstantiated actions, we may find **potential** threats
    - stack(B,y) **may** threaten the causal link, but only if $x = y$
    - Can be resolved by adding a constraint: $x \neq y$



Complete Example

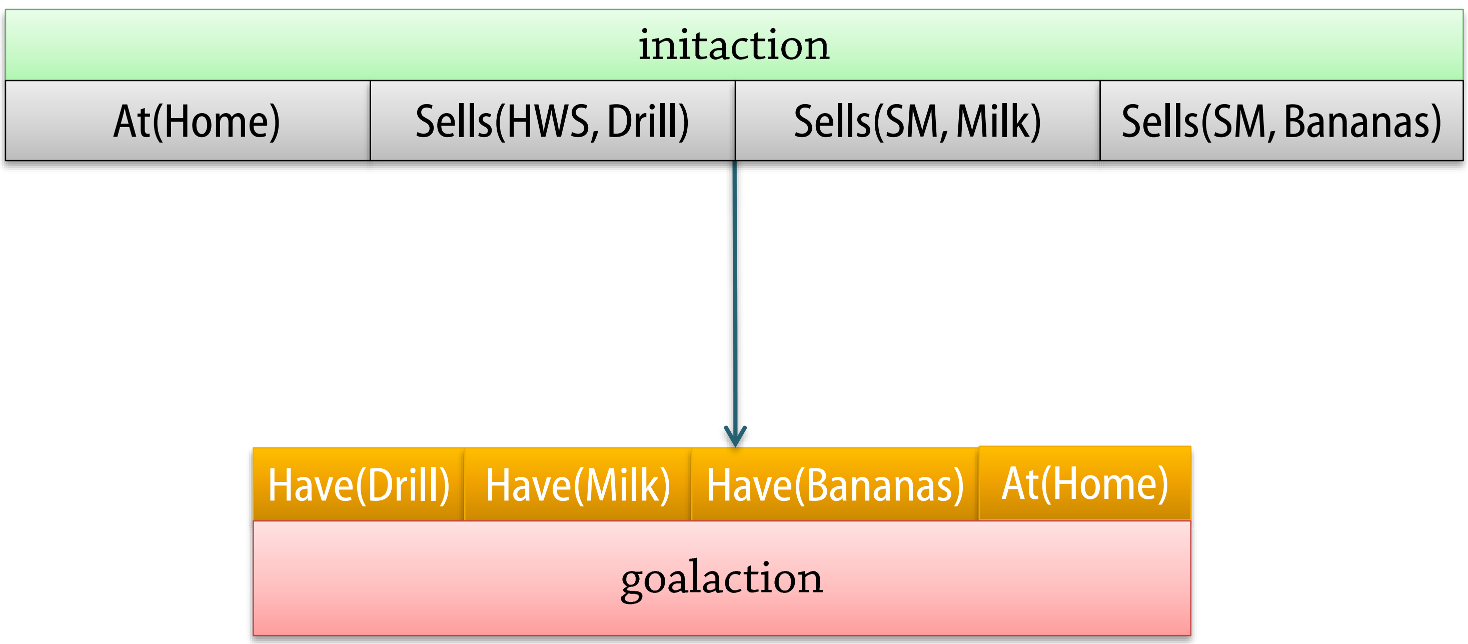
Example

- Running Example: Similar to an example in AIMA
  - Russell and Norvig's *Artificial Intelligence: A Modern Approach* (1st ed.)

  - Operator **Go(from,to)**
    - Precond: At(from)
    - Effects: At(to), ¬At(from)
  - Operator **Buy(product, store)**
    - Precond: At(store), Sells(store, product)
    - Effects: Have(product)
  - **Initial state**
    - At(Home), Sells(HWS, Drill), Sells(SM, Milk), Sells(SM, Bananas)
  - **Goal**
    - At(Home), Have(Drill), Have(Milk), Have(Bananas)

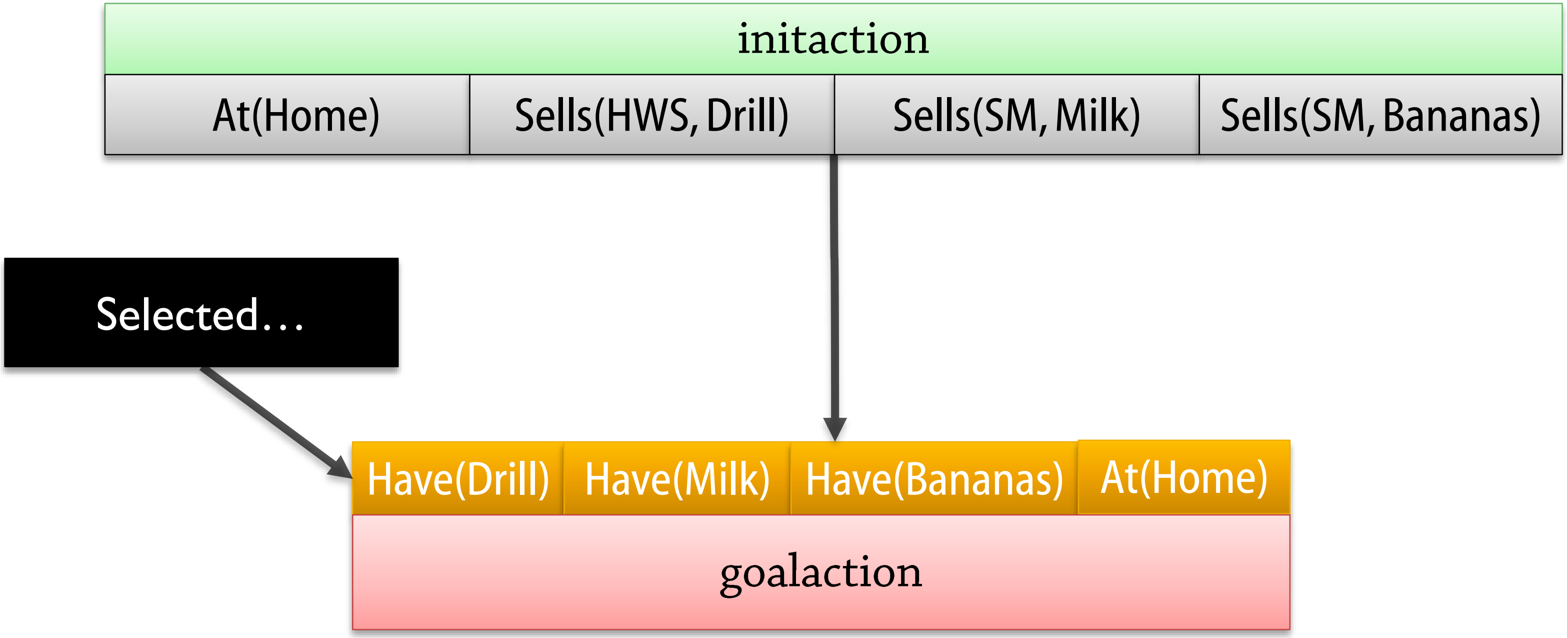
Example (continued)

- Initial plan: **initaction**, **goalaction**, and a precedence constraint



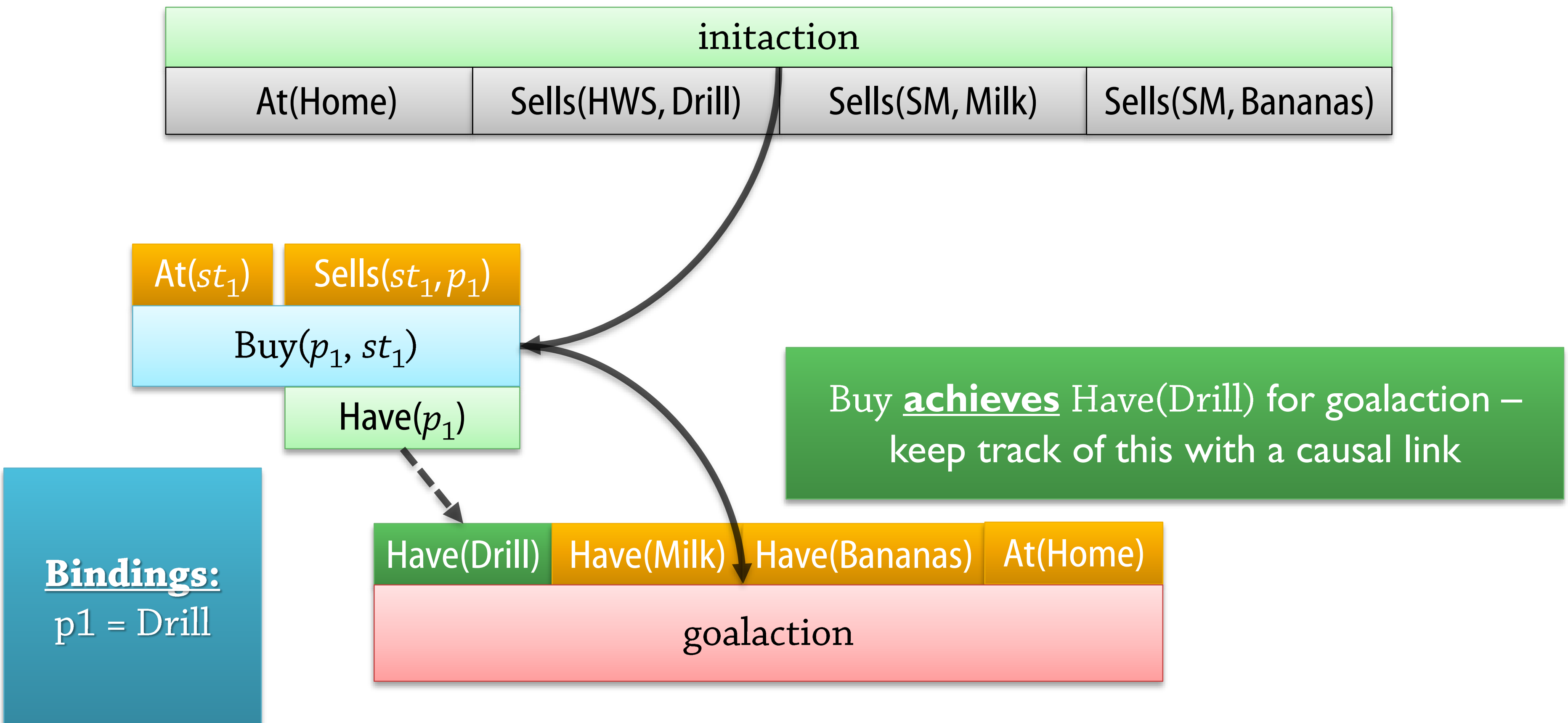
Example (continued)

- Four **flaws** exist: Open goals
  - Suppose our heuristics tell us to resolve **Have(Drill)** first



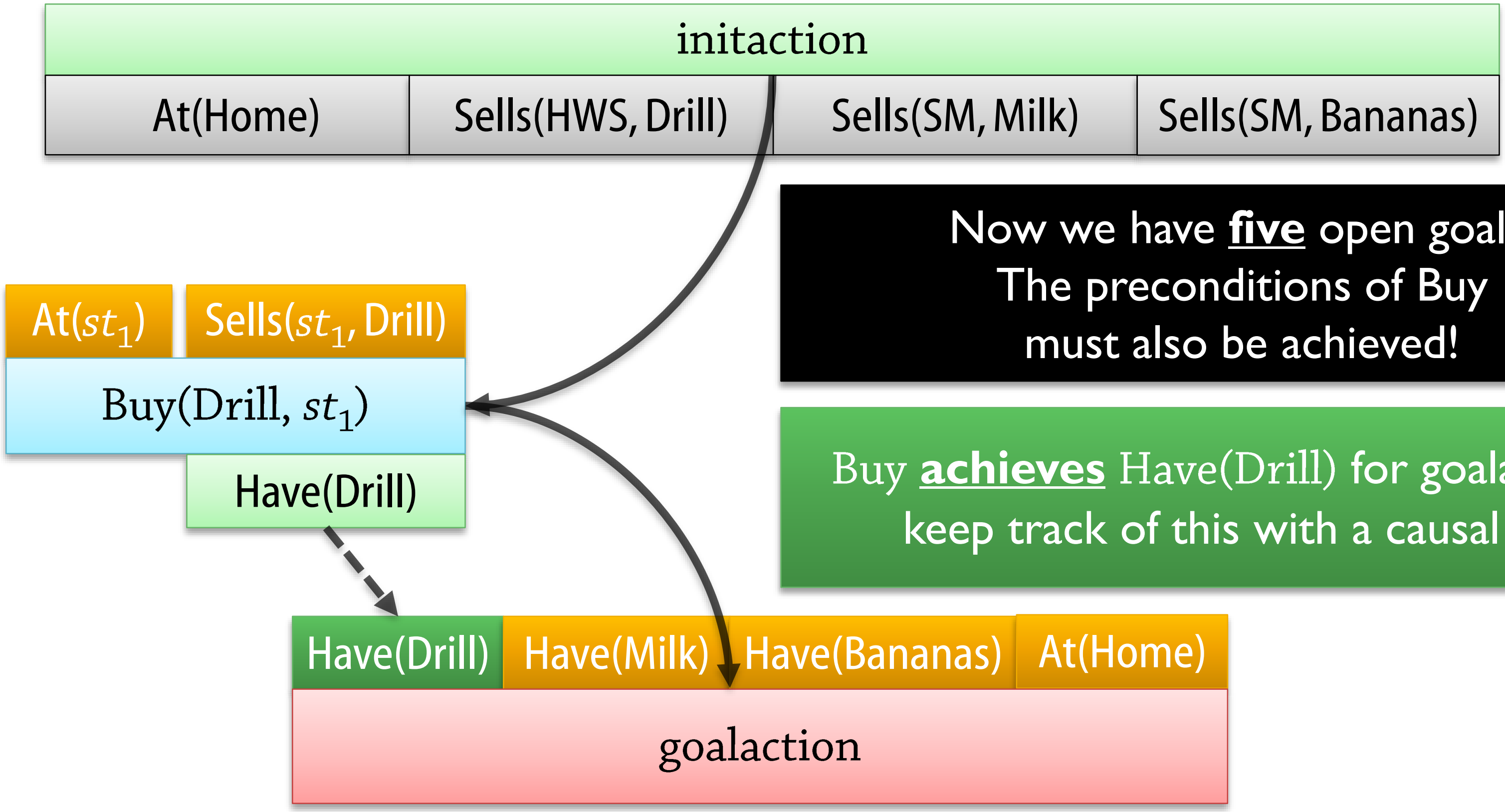
Example (continued)

- Have(drill) is not achieved by any action in the current plan
- But **Buy(product, store)** achieves Have(product)
  - Partially instantiate:
    Buy(Drill, store) (right now we don't care **where** we buy it)



Example (continued)

- **Alternative Notation** for simplicity
  - Variable bindings are implicit in the diagram

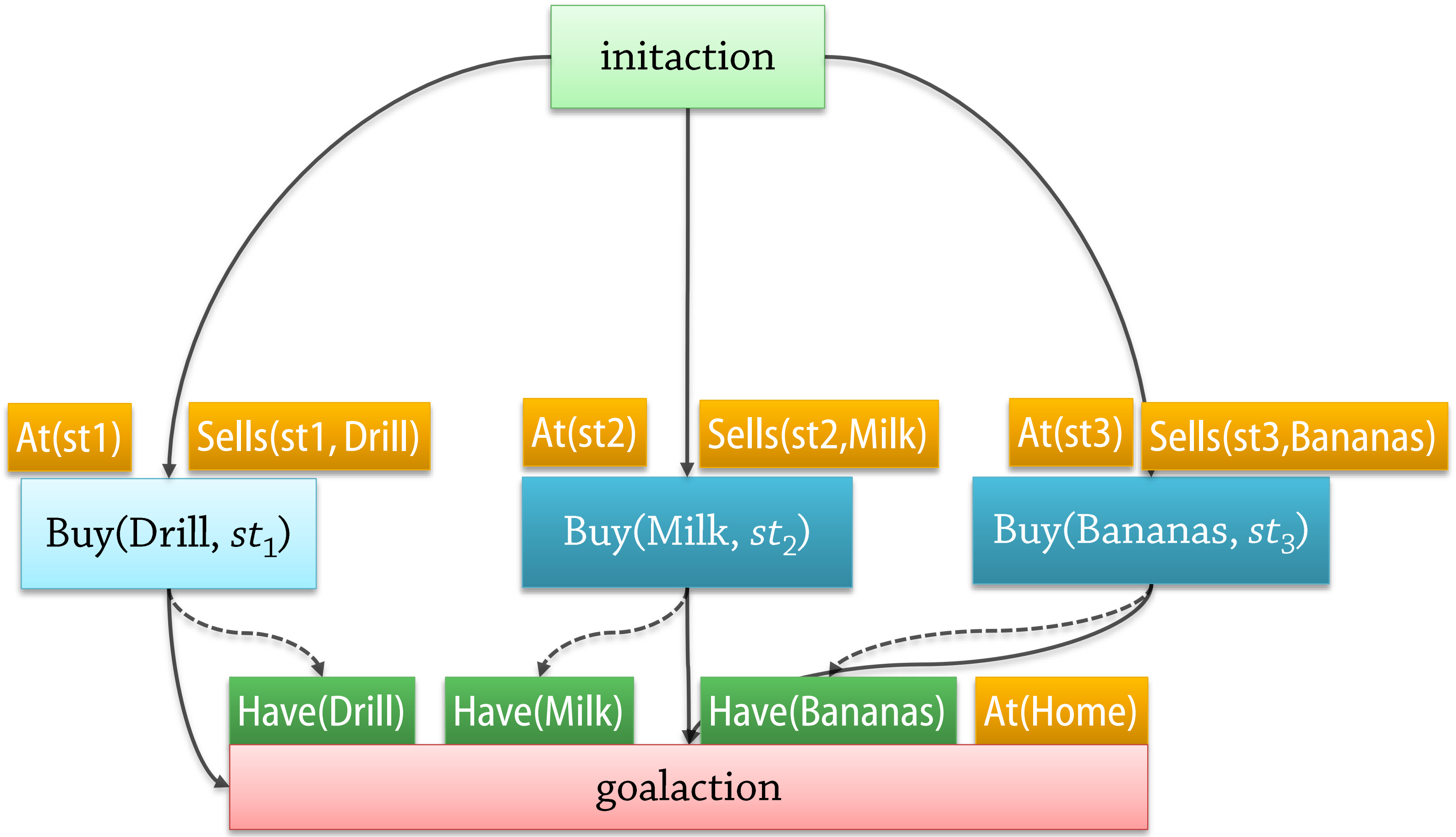


Now we have **five** open goals:
The preconditions of Buy
must also be achieved!

Buy **achieves** Have(Drill) for goalaction – keep track of this with a causal link

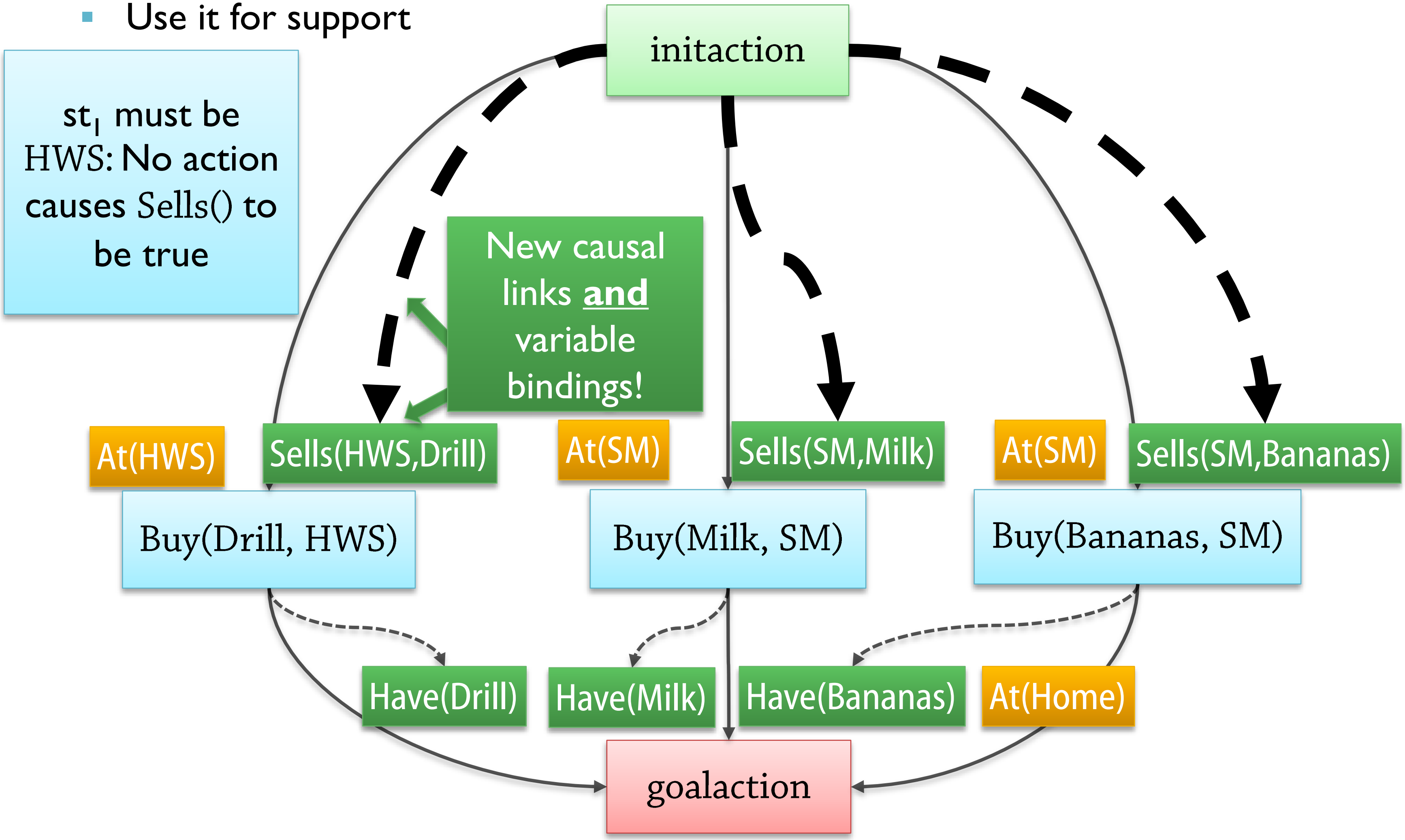
Example (continued)

- The first **three** refinement steps
  - These are the only possible ways to establish the Have preconditions
  - We don't care in which **order** we buy things!



Example (continued)

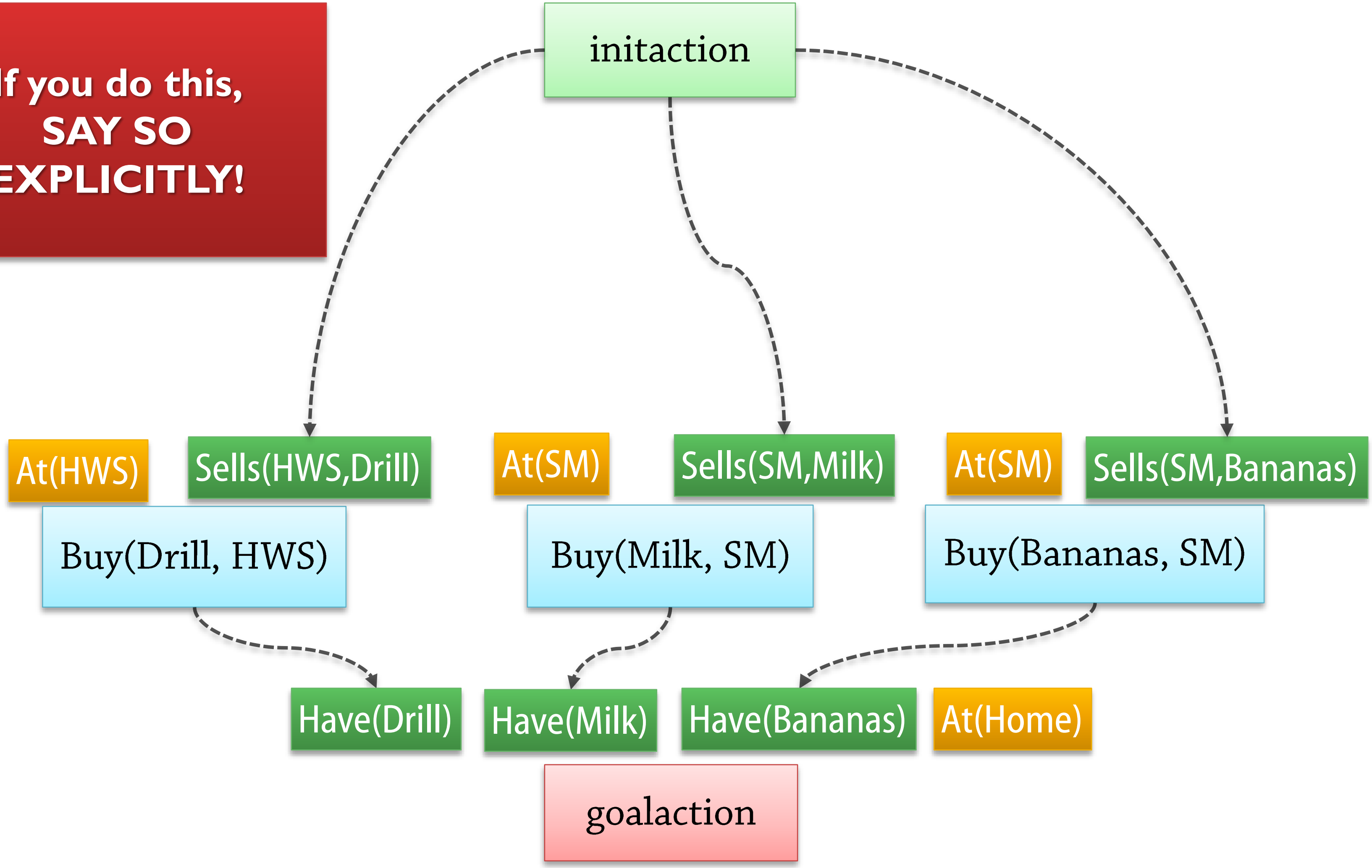
- Three more refinement steps
  - No action causes Sells(…) to be true – except the "fake" initial action!
  - Use it for support



Example (continued)

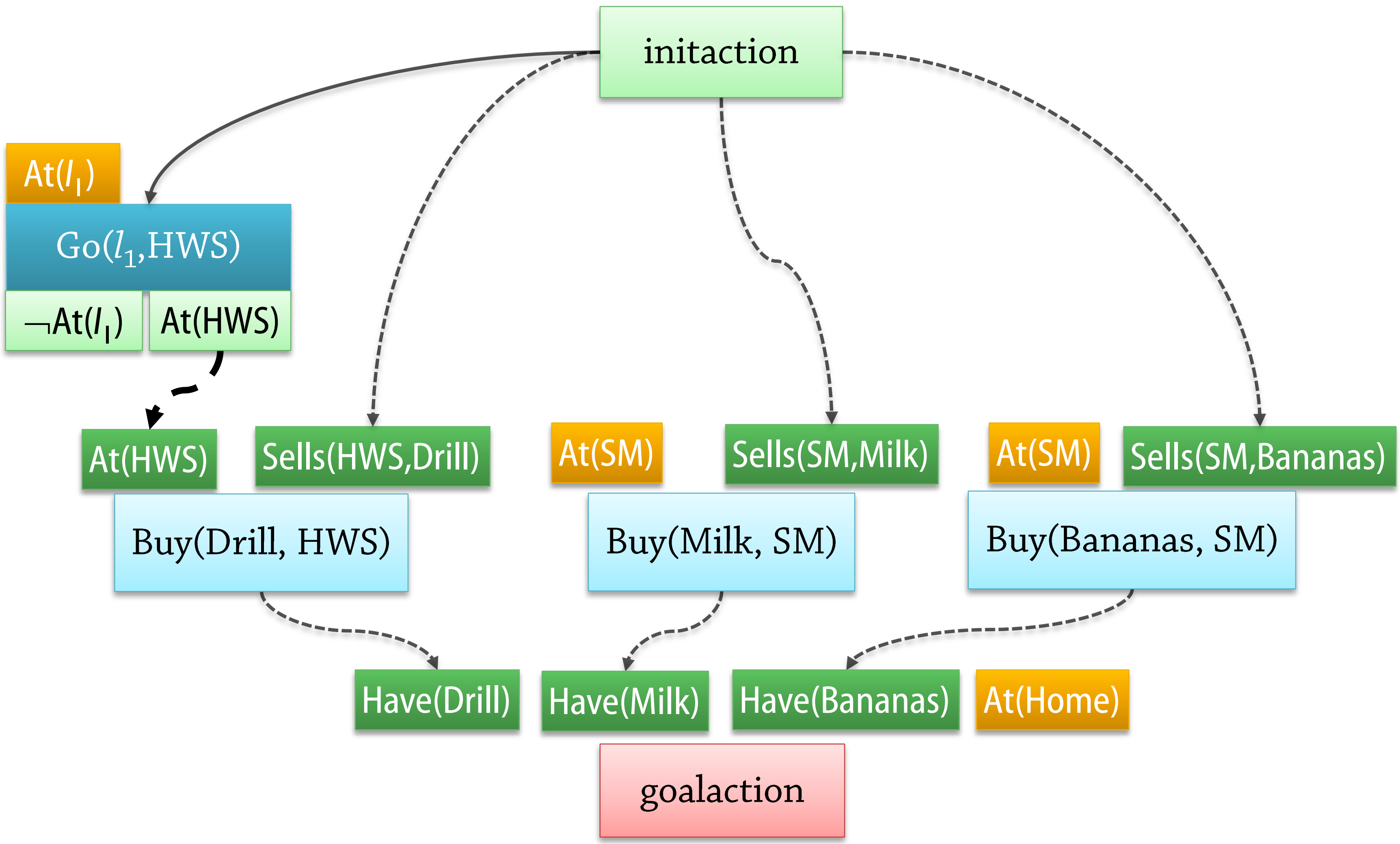
- It's getting messy!
  - Let's omit the precedence constraints that are implicit in causal links…

If you do this, SAY SO EXPLICITLY!



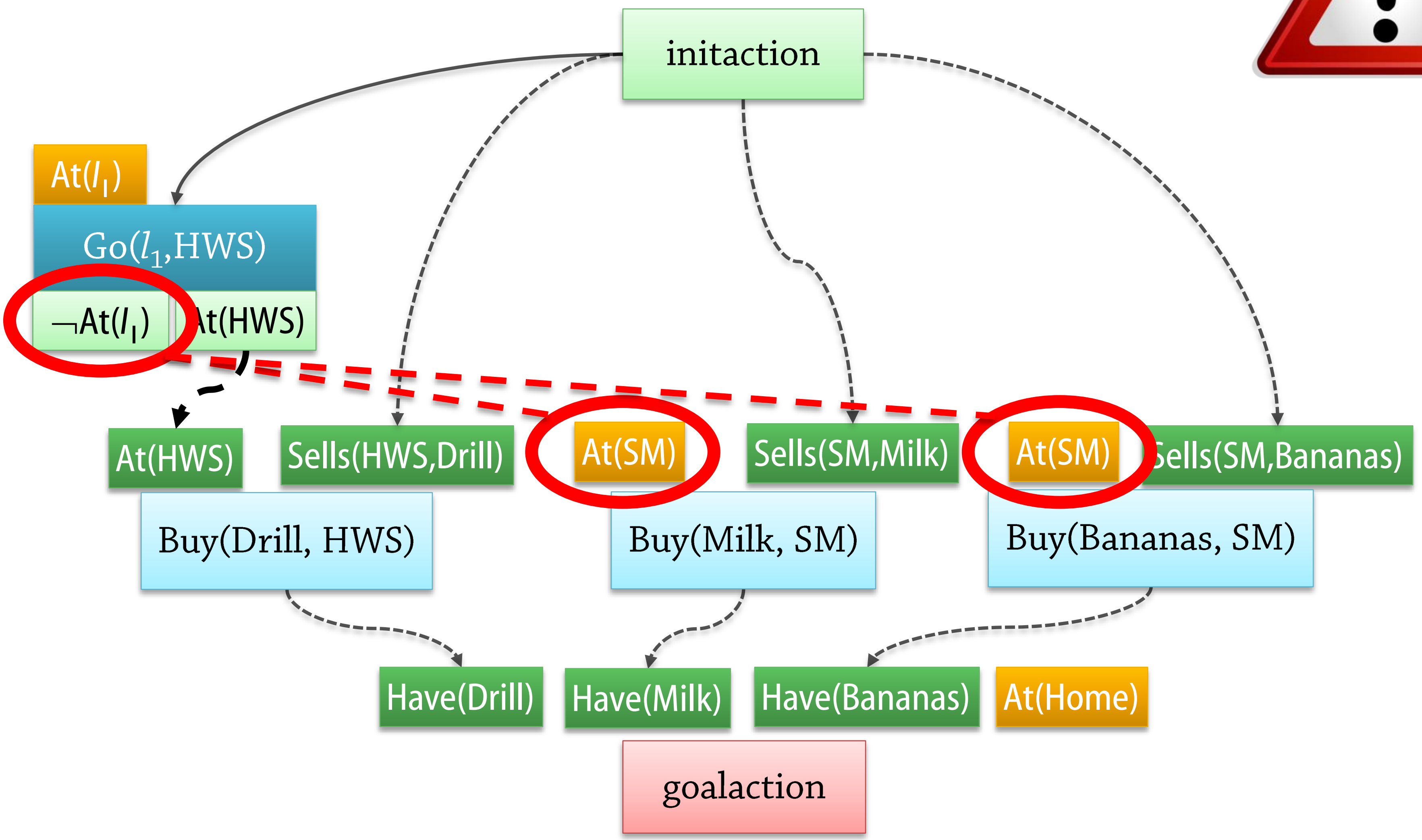
Example (continued)

- To establish At(HWS): **Must** go there from **somewhere**



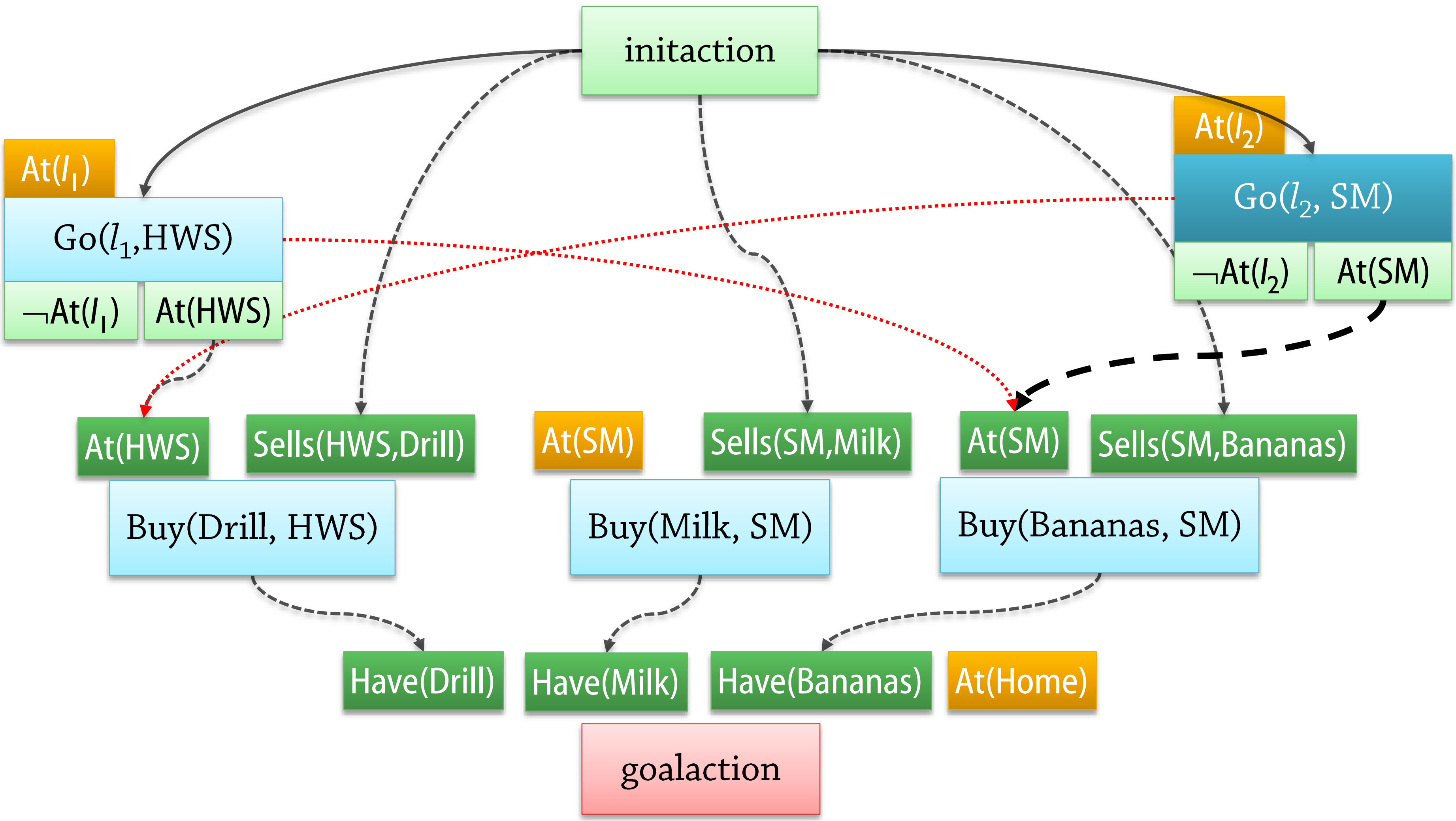
Example (continued)

- Does $\neg at(l_1)$ threaten At(SM)?
  - No! Only a **causal link** to At(SM) can be threatened



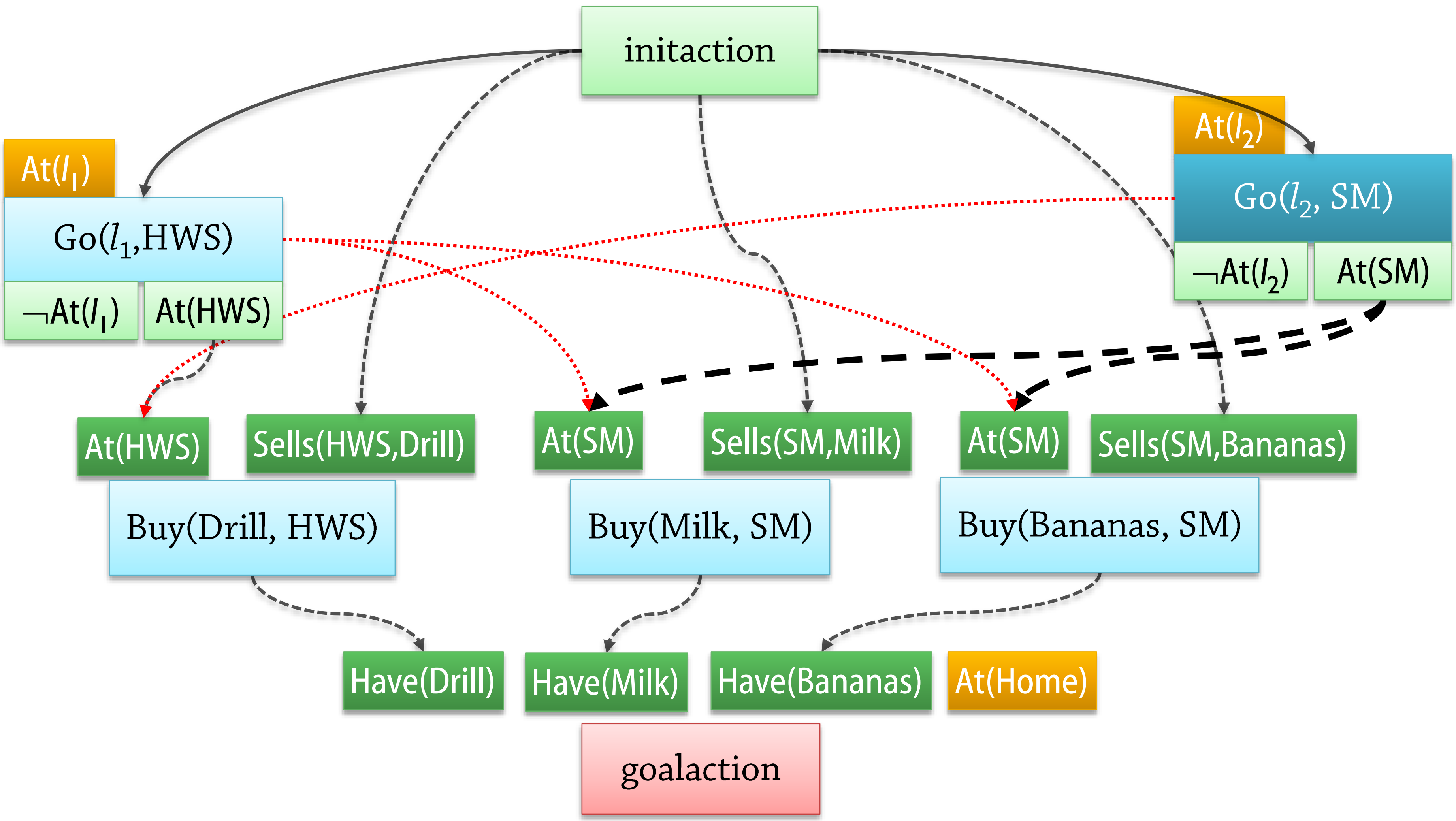
Example (continued)

- To establish At(SM): Must go there from **somewhere**
  - Mutual **threats**…



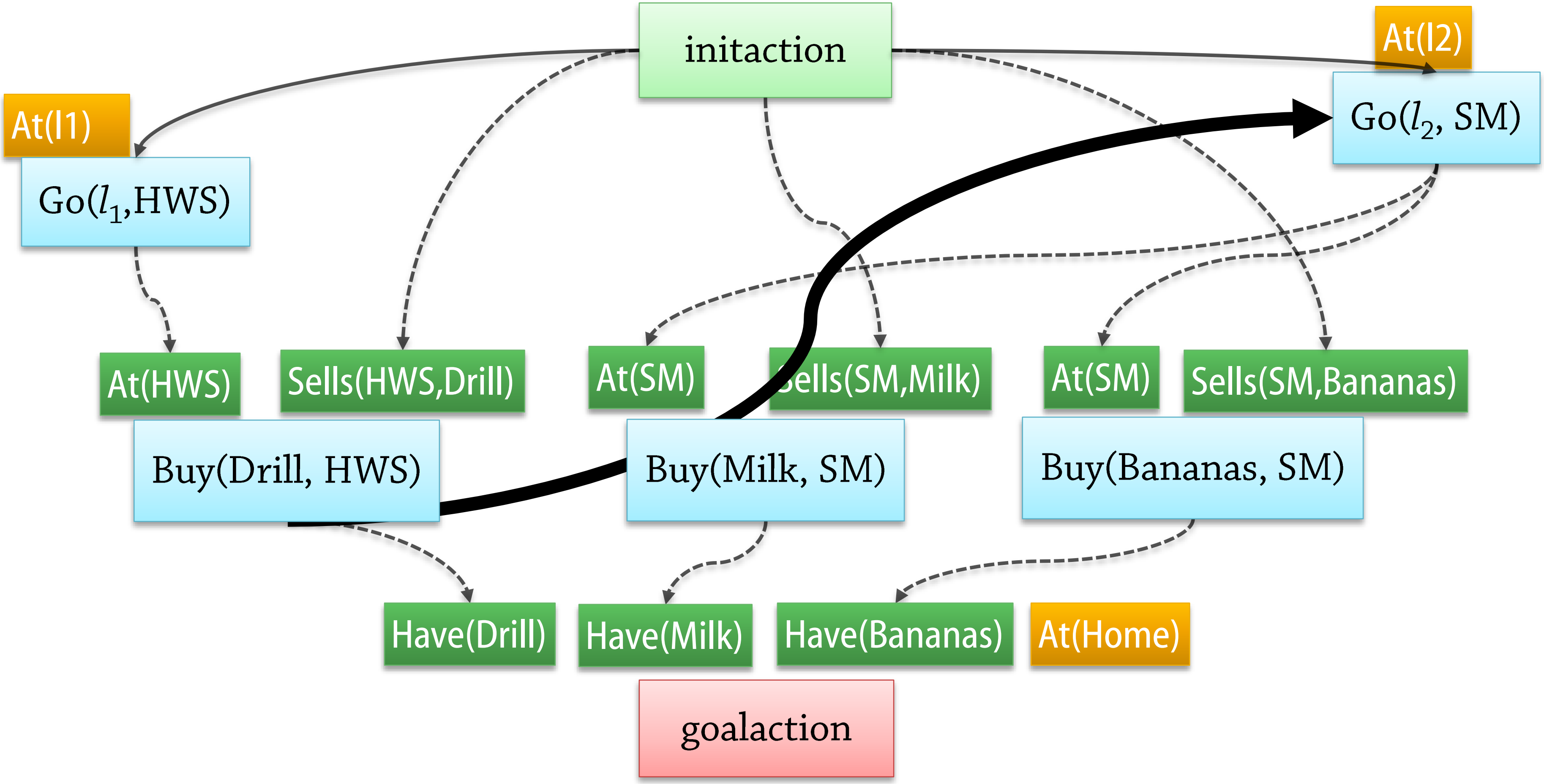
Example (continued)

- Let's use the same action for **both** At(SM) preconditions…
  - More threats – could deal with them now or wait



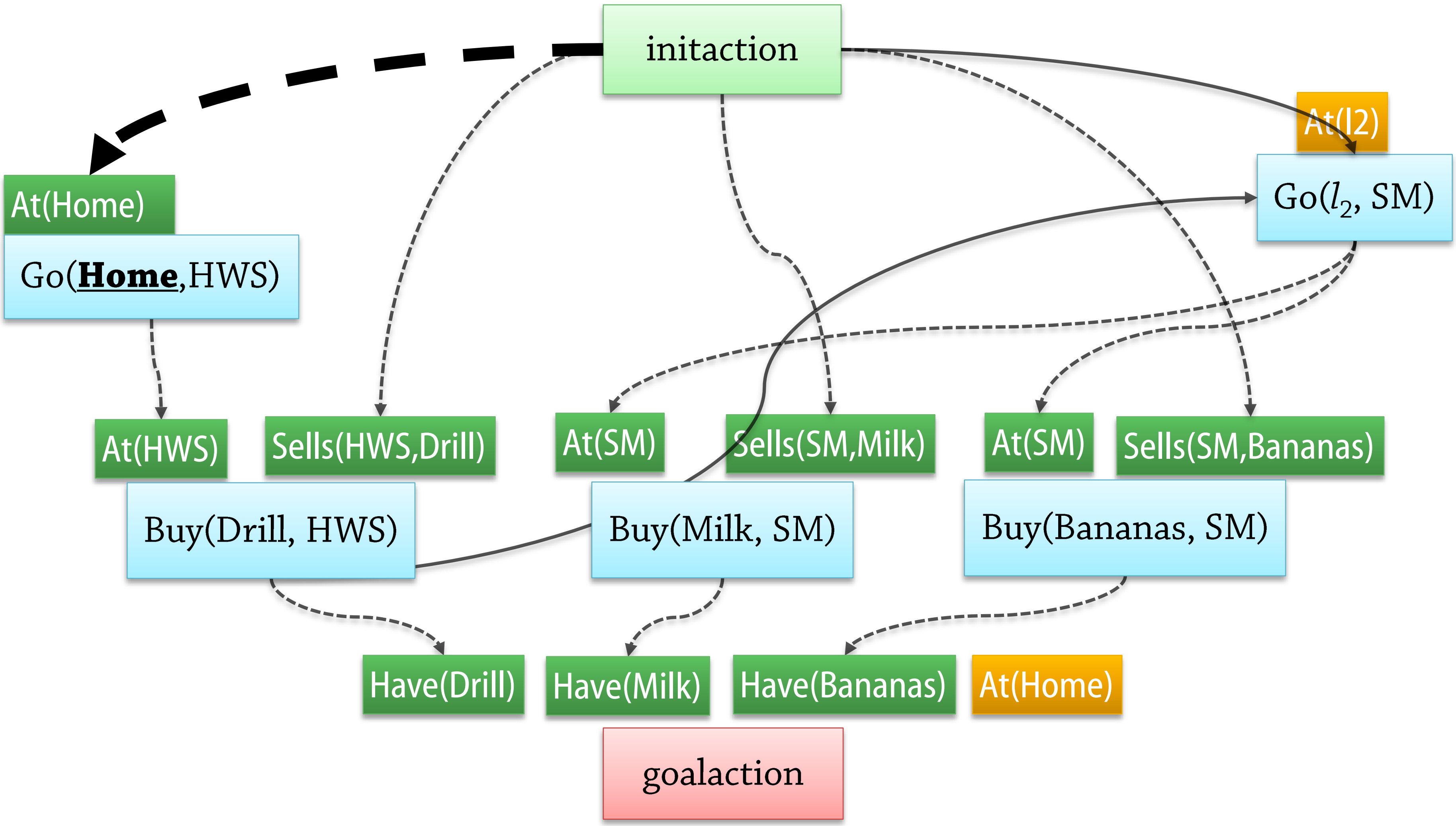
Example (continued)

- **Nondet. choice**: how to resolve the threat to At(HWS)?
  - Our choice: make the "requirer" Buy(Drill) precede the "threatener" Go(l2, SM)
  - Also happens to resolve the other two threats
    - "Threatener" Go(l1, HWS) before "achiever" Go(l2, SM)



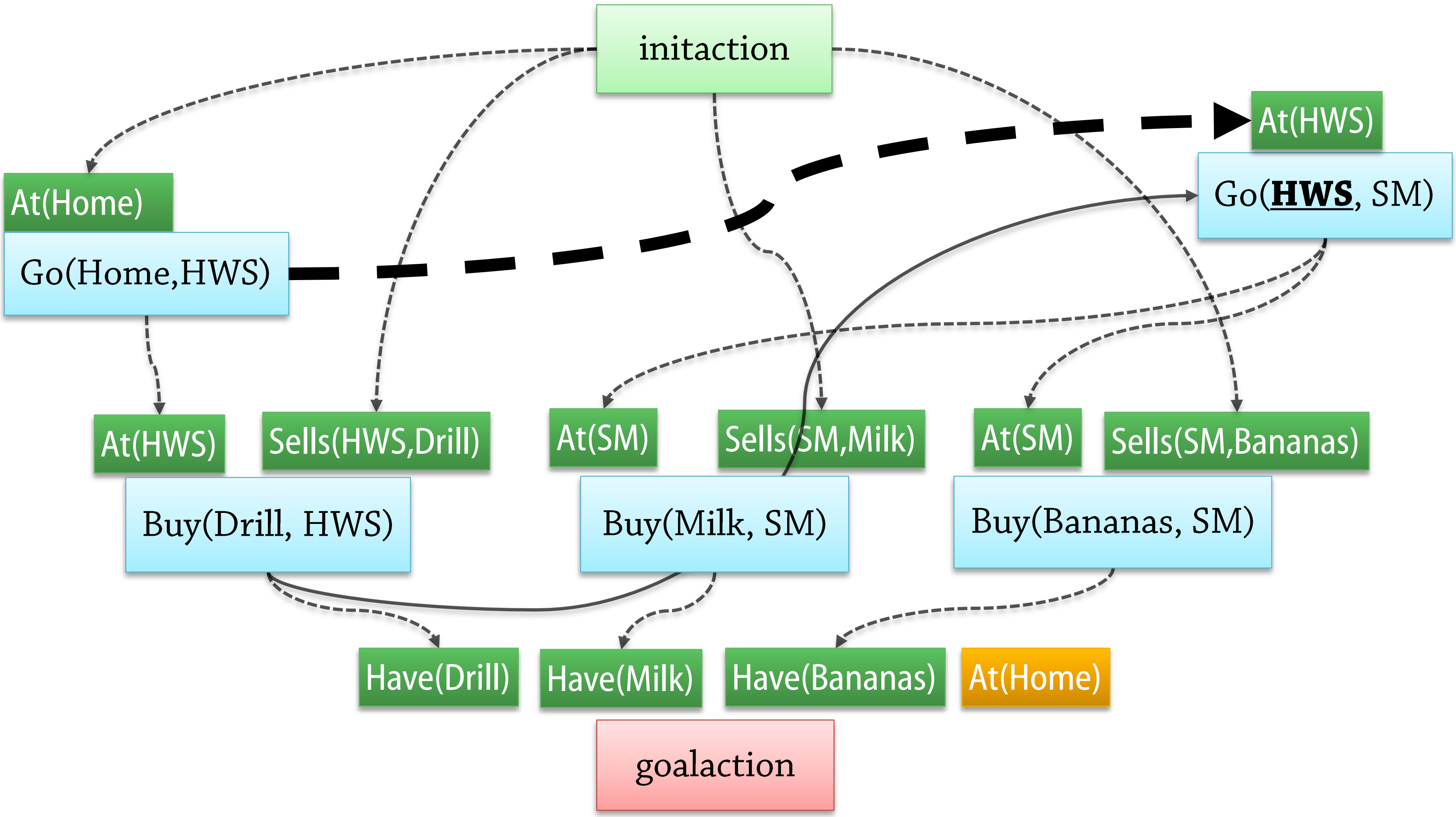
Example (continued)

- **Nondet. choice**: how to establish At(l_1)?
  - We'll do it from initaction, with l_1=Home



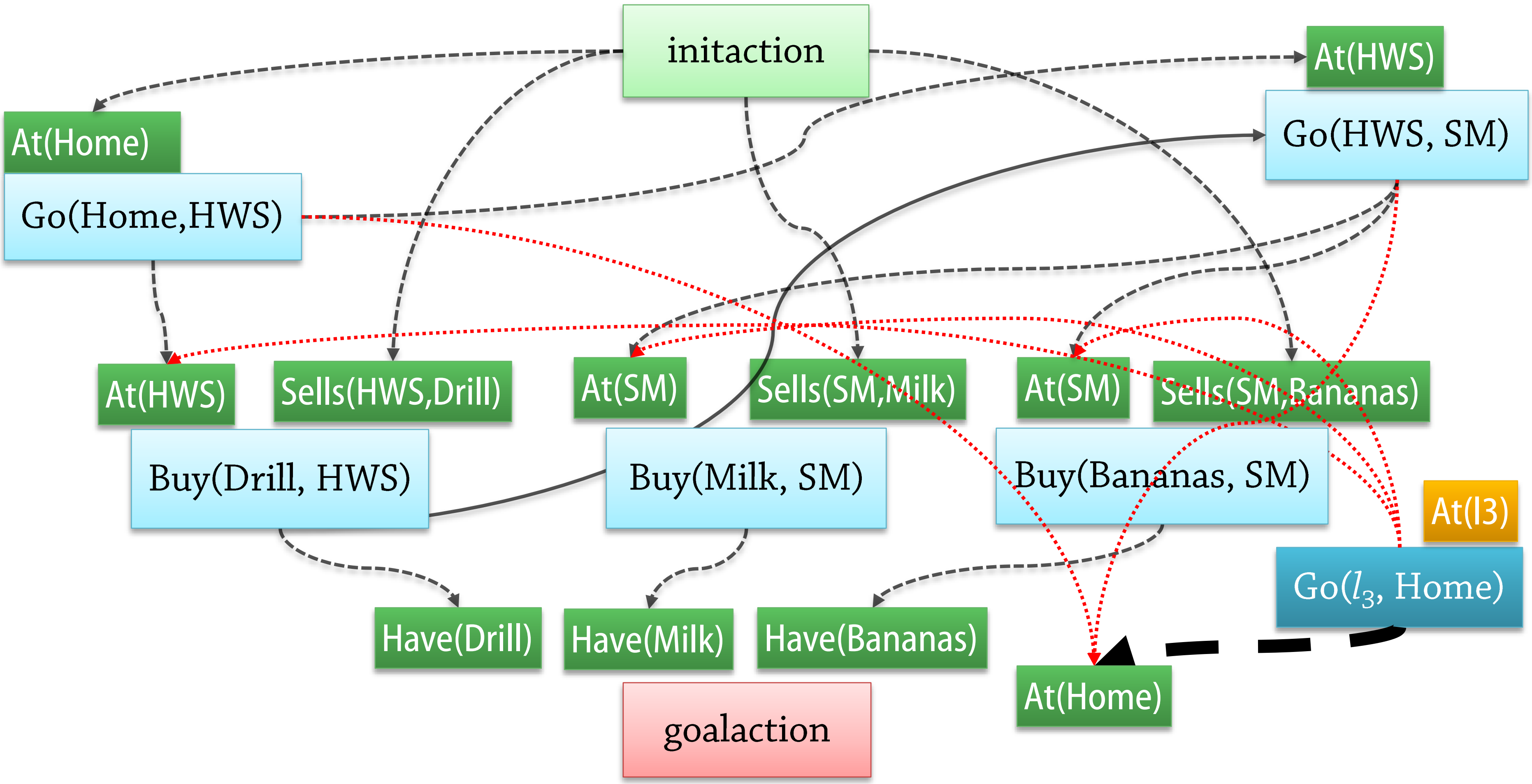
Example (continued)

- **Nondeterministic choice**: how to establish At(l_2)?
  - We'll do it from Go(Home,HWS), with l_2= HWS



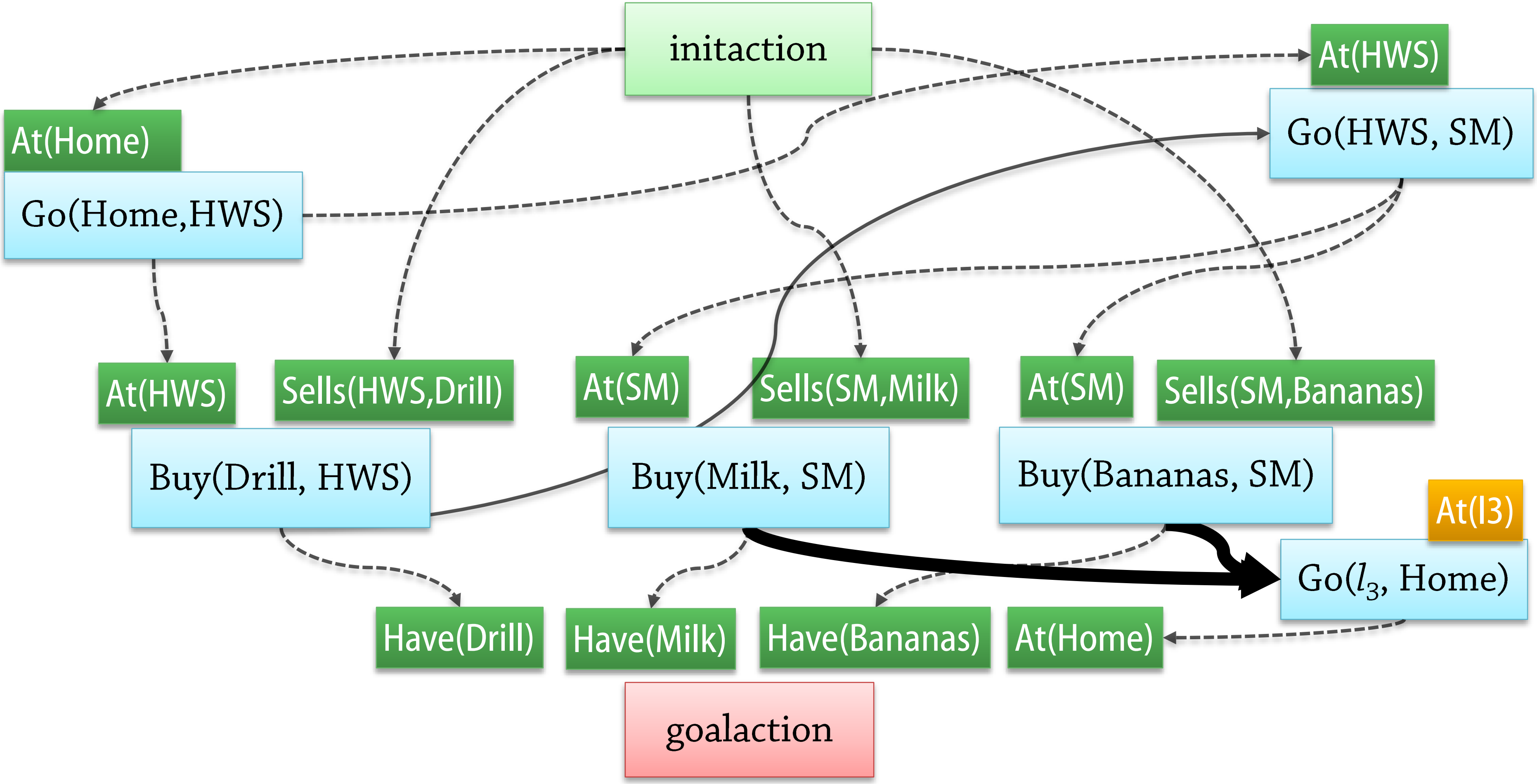
Example (continued)

- The only possible way to establish At(Home) for goalaction
  - This creates several new threats



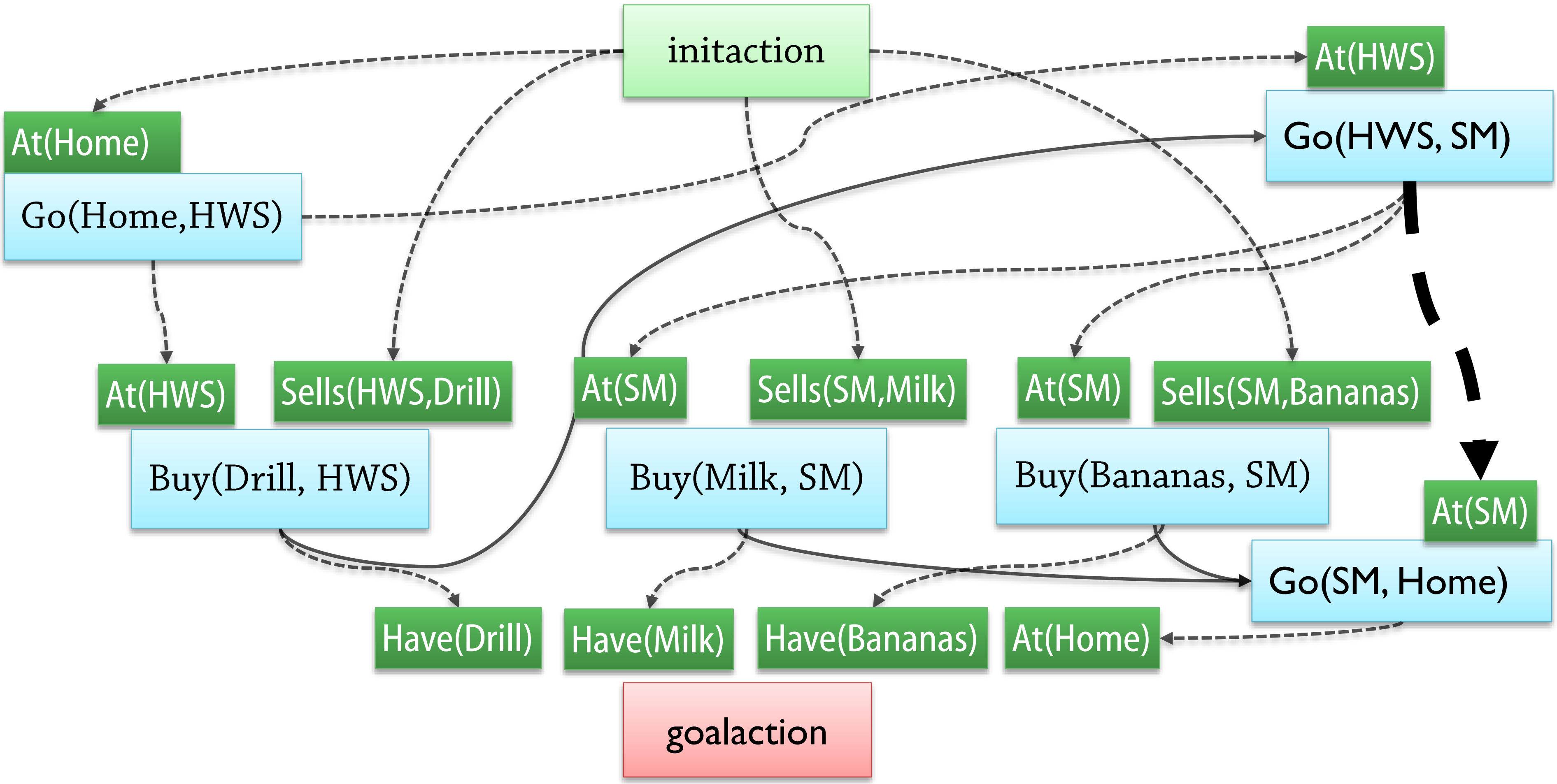
Example (continued)

- To remove the threats to At(SM) and At(HWS):
  - Make go(HWS,SM) and go(Home,HWS) precede Go(l_3,Home)
  - This also removes the other threats



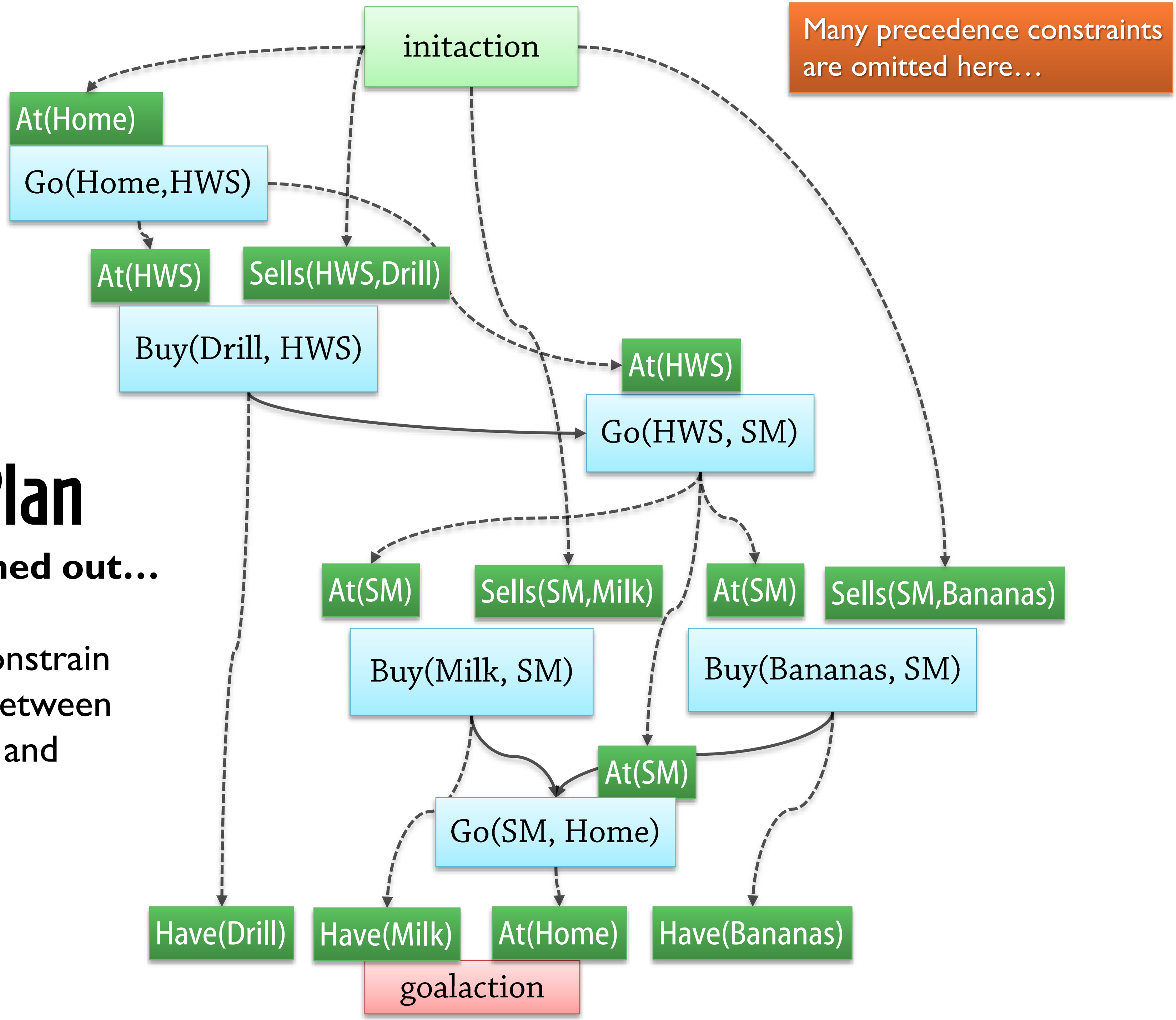
Final Plan

- Establish $\mathrm{At}(l_3)$ with $l_3 = \mathrm{SM}$

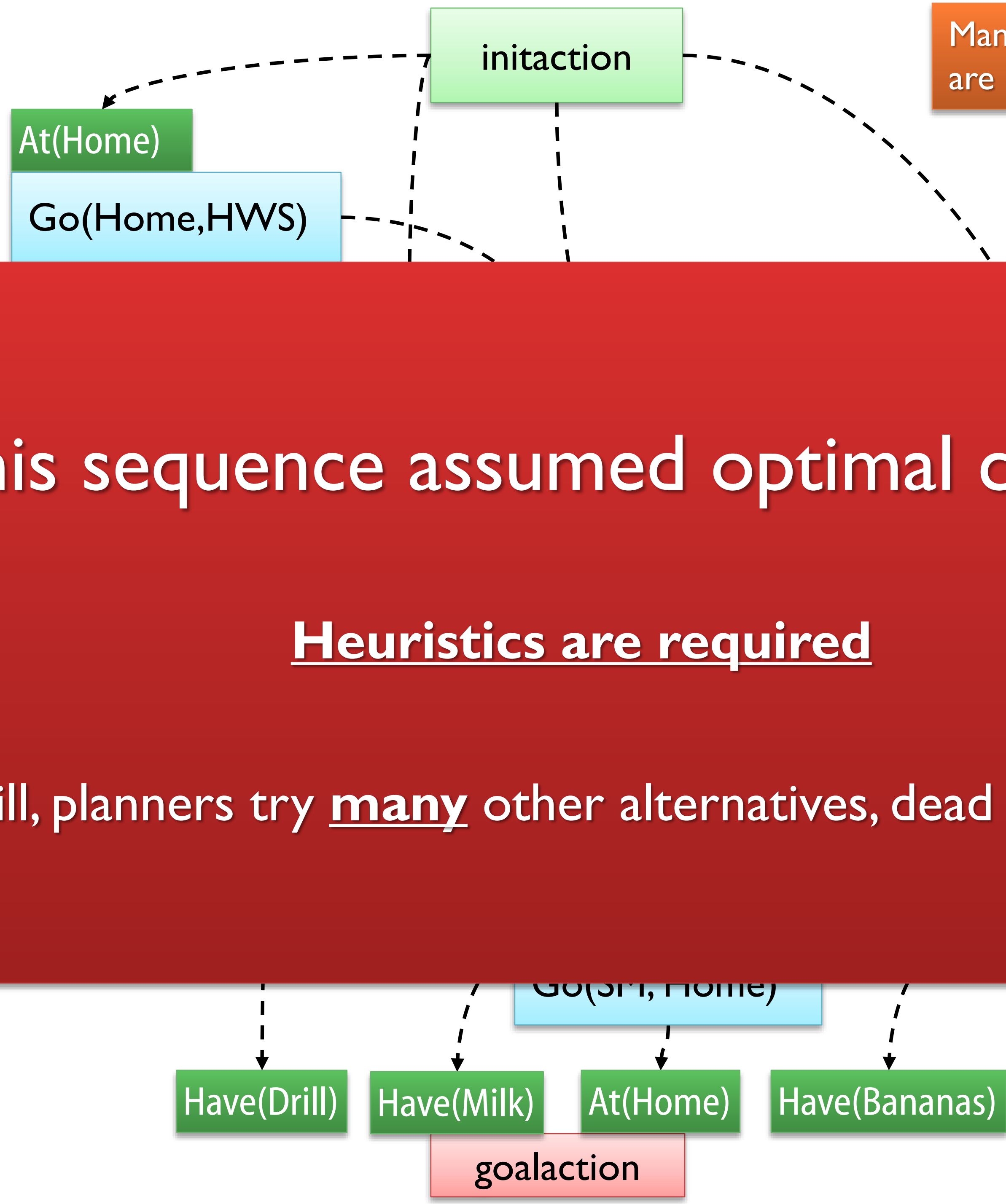


Final Plan

Straightened out…
(Note: Still does not constrain the order between buying milk and bananas)



Many precedence constraints are omitted here…



This sequence assumed optimal choices!

<u>Heuristics are required</u>

Still, planners try **<u>many</u>** other alternatives, dead ends, etc.