Graph-based Planning

Slides based on material from: Prof. Maria Fox

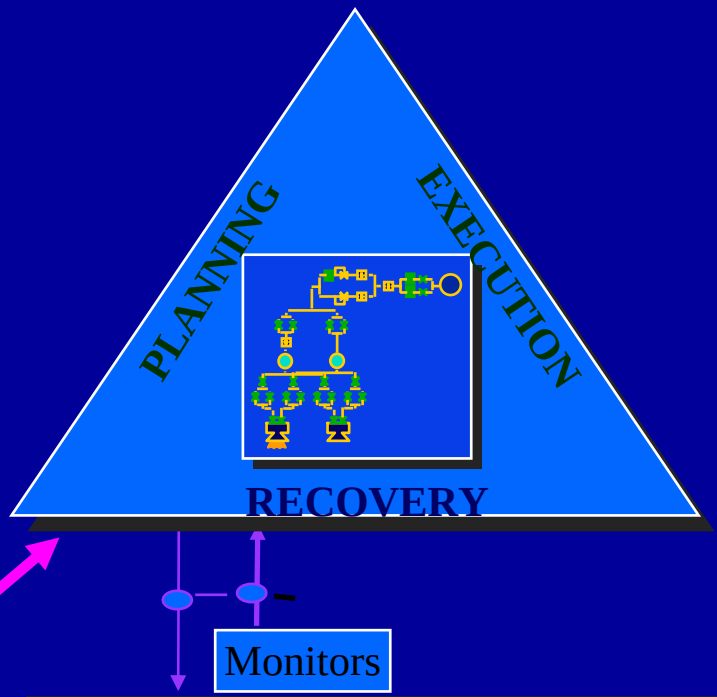
Brian C. Williams

October 25th & 27th, 2003

16.410 - 13

Autonomous Agents

Self-commanding
Self-diagnosing
Self-repairing



Commanded at:
- Mission level
- Engineering level

Command dispatch
Fault protection
Attitude control

Mission Goal Scenario

Reading and Assignment for Planning Lectures

- Graph-based Planning
  AIMA Chapter 11

- AIMA = “Artificial Intelligence: A Modern Approach,” by Russell and Norvig.

- Problem Set #6
  - Out Friday, Oct 22nd
  - Due Friday, Oct 29th.

Outline

- Operator-based Planning
- Graph Plan
  - The Graph Plan Planning Problem
  - Graph Construction
  - Solution Extraction
  - Properties
  - Termination with Failure

Operator-based Planning Problem

- *State*
  - A consistent conjunction of propositions (positive literals)
    - E.g., (and (cleanhands) (quiet) (dinner) (present) (noGarbage))
    - All unspecified propositions are false
- *Initial State*
  - Problem state at time i = 0
    - E.g., (and (cleanHands) (quiet))
- *Goal State*
  - A partial state
    - E.g., (and (noGarbage) (dinner) (present))
  - The planner must put the system in a final state that satisfies the conjunction.

Example: Dinner Date Problem

Initial Conditions: (and (cleanHands) (quiet))

Goal: (and (noGarbage) (dinner) (present))

Actions:

(:operator **carry** *:precondition*
:effect (and (noGarbage) (not (cleanHands)))

(:operator **dolly** *:precondition*
:effect (and (noGarbage) (not (quiet)))

(:operator **cook** *:precondition* (cleanHands)
:effect (dinner))

(:operator **wrap** *:precondition* (quiet)
:effect (present))

+ noops

(Parameterized) Operator Schemata

- Instead of defining:

  pickup-A and **pickup-B** and …

- Define a schema:

(:operator **pick-up**
:***parameters*** **((block ob1))**
:*precondition* (and (clear ob1)
(on-table ob1)
(arm-empty))
:effect (and (not (clear ob1))
(not (on-table ob1))
(not (arm-empty))
(holding ob1)))

Note: strips doesn't allow derived effects; you must be complete!

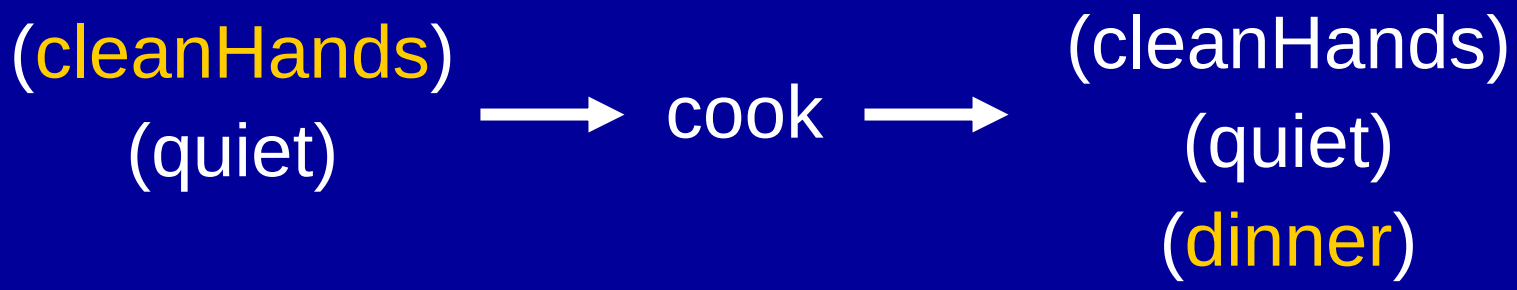
?var denotes a free variable

Operator Execution at Time i

If all propositions of :precondition appear in state i,
Then create state i+1 from i, by

- adding to i, all “add” propositions in :effects,
- removing from i, all “delete” propositions in :effects.

(:operator **cook** *:precondition* (cleanHands)
:effect (dinner))



Operator Execution at Time i

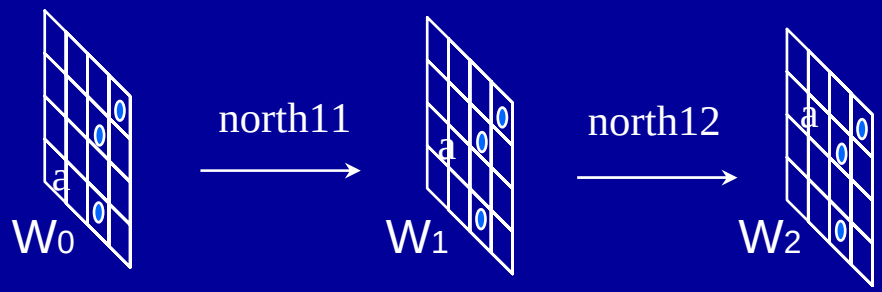
If all propositions of :precondition appear in state i,
Then create state i+1 from i, by

- adding to i, all “add” propositions in :effects,
- removing from i, all “delete” propositions in :effects.

(:operator **dolly** *:precondition*
:effect (and (noGarbage) (not (quiet)))

Operator-based Planning Problems

- Input
  - Set of world states
  - Action operators
    - Fn: world-state→world-state
  - Initial state of world
  - Goal
    - partial state
      (set of world states)
- Output
  - Sequence of actions
    - Complete: Achieve goals
    - Consistent: No negative side-effects



Operator-based Planning Problem

(:operator **carry**
:precondition
:effect (:and (noGarbage) (not (cleanHands)))

Preconditions: Propositions that must be true to apply the operator.

- A conjunction of propositions (no negated propositions).

Effects: Propositions that the operator changes, given the preconditions are satisfied.

- A conjunction of propositions (called adds) and their negation (called deletes).

What Assumptions are Implied?

```
(:operator pick-up
          :parameters ((block ob1))
          :precondition (and (clear ob1)
                             (on-table ob1)
                             (arm-empty))
          :effect (and (not (clear ob1))
                       (not (on-table ob1))
                       (not (arm-empty))
                       (holding ob1)))
```

- Atomic time.
- Agent is omniscient (no sensing necessary).
- Agent is sole cause of change.
- Actions have deterministic effects.
- No indirect effects.

⇨ STRIPS Assumptions

Outline

- Operator-based Planning
- Graph Plan
  - The Graph Plan Solutions
  - Graph Construction
  - Solution Extraction
  - Properties
  - Termination with Failure

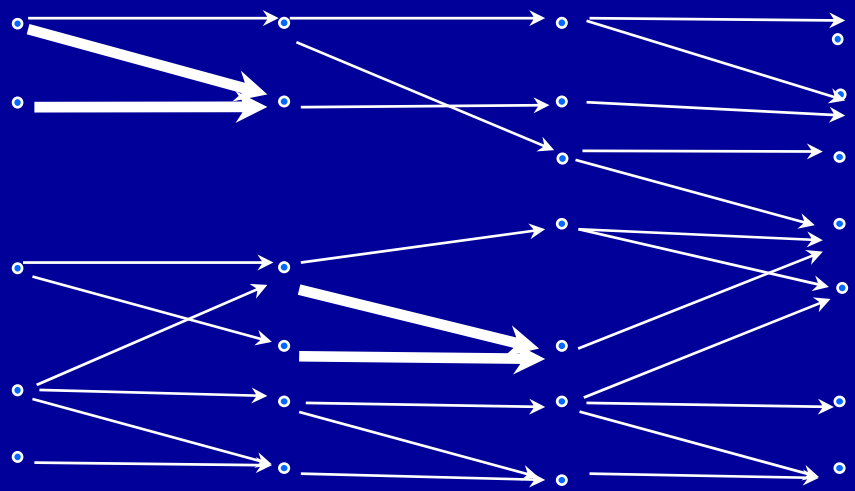
Graph Plan

- Graphplan was developed in 1995 by Avrim Blum and Merrick Furst, at CMU.

- Graphplan approach extended to reason with temporally extended actions, metric and non-atomic preconditions and effects.

Approach: Graph Plan

1. Constructs compact constraint encoding of state space from operators and initial state, which prunes many invalid plans – *Plan Graph.*
2. Generates plan by searching for a consistent subgraph that achieves the goals.



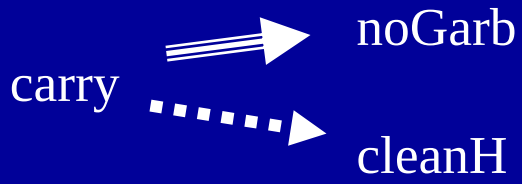
Outline

- Operator-based Planning
- Graph Plan
  - The Graph Plan Planning Problem
  - Graph Construction
  - Solution Extraction
  - Properties
  - Termination with Failure

Visualizing Actions in a Plan Graph

(:operator **cook** *:precondition* (cleanHands)
:effect (dinner))

(:operator **carry** *:precondition*
:effect (:and (noGarbage) (not (cleanHands)))



Visualizing Actions in a Plan Graph

- ***Persistence actions (Noops)***
  - Every literal has a no-op action, which maintains it from time i to i+1.

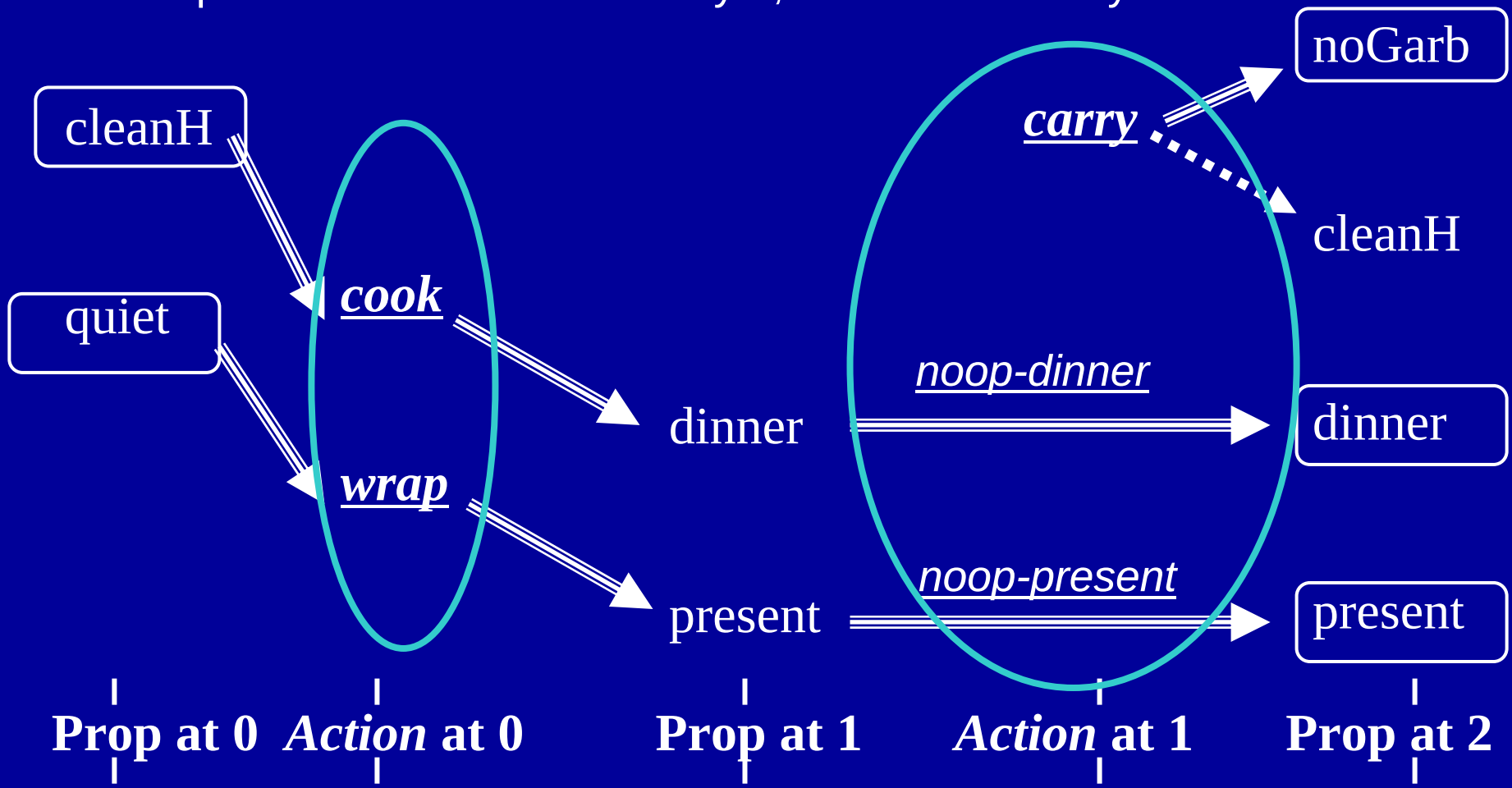
(:operator noop-P *:precondition* (P) *:effect* (P))

In Blum & Furst: (& lecture) Only persist positive literals .

AIMA: Persists negative literals as well.

A Plan in GraphPlan <Actions[i] >

- Sets of concurrent actions performed at each time [i]
  - Concurrent actions can be interleaved in any order.
    - ⇨If actions a and b occur at time i, then it must be valid to perform either a followed by b, OR b followed by a.



A Complete Consistent Plan

Given that the initial state holds at time 0,
a plan is a solution iff:

- Complete:
  - The goal propositions all hold in the final state.

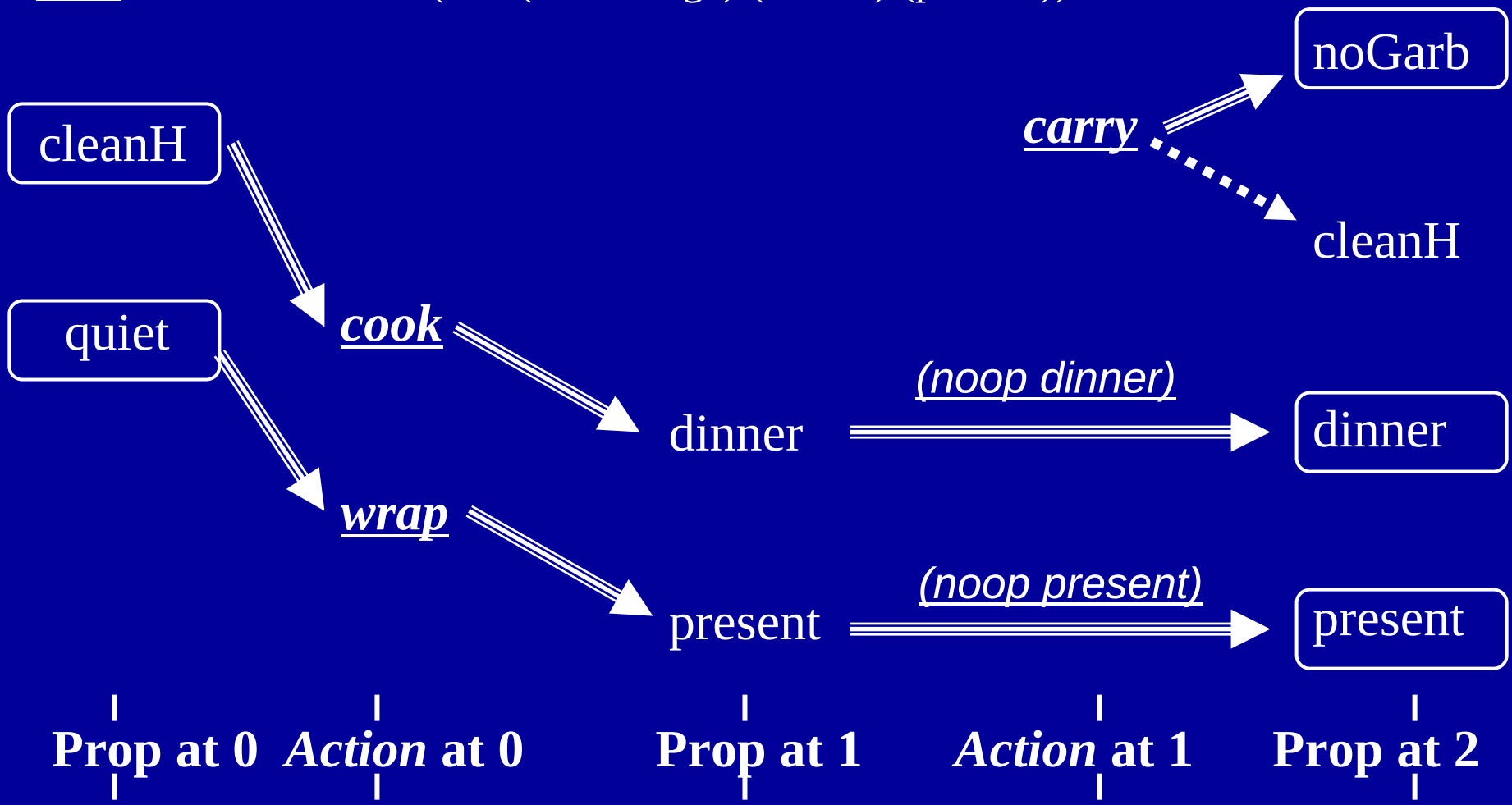
  - The preconditions of every operator at time i,
    is satisfied by a proposition at time i.

- Consistent:
  - The operators at any time i can be executed in any order, without one of these operators undoing:
    - the preconditions of another operator at time i.
    - the effects of another operator at time i.

Example of a Complete Consistent Plan

Initial Conditions: (and (cleanHands) (quiet))

Goal: (and (noGarbage) (dinner) (present))



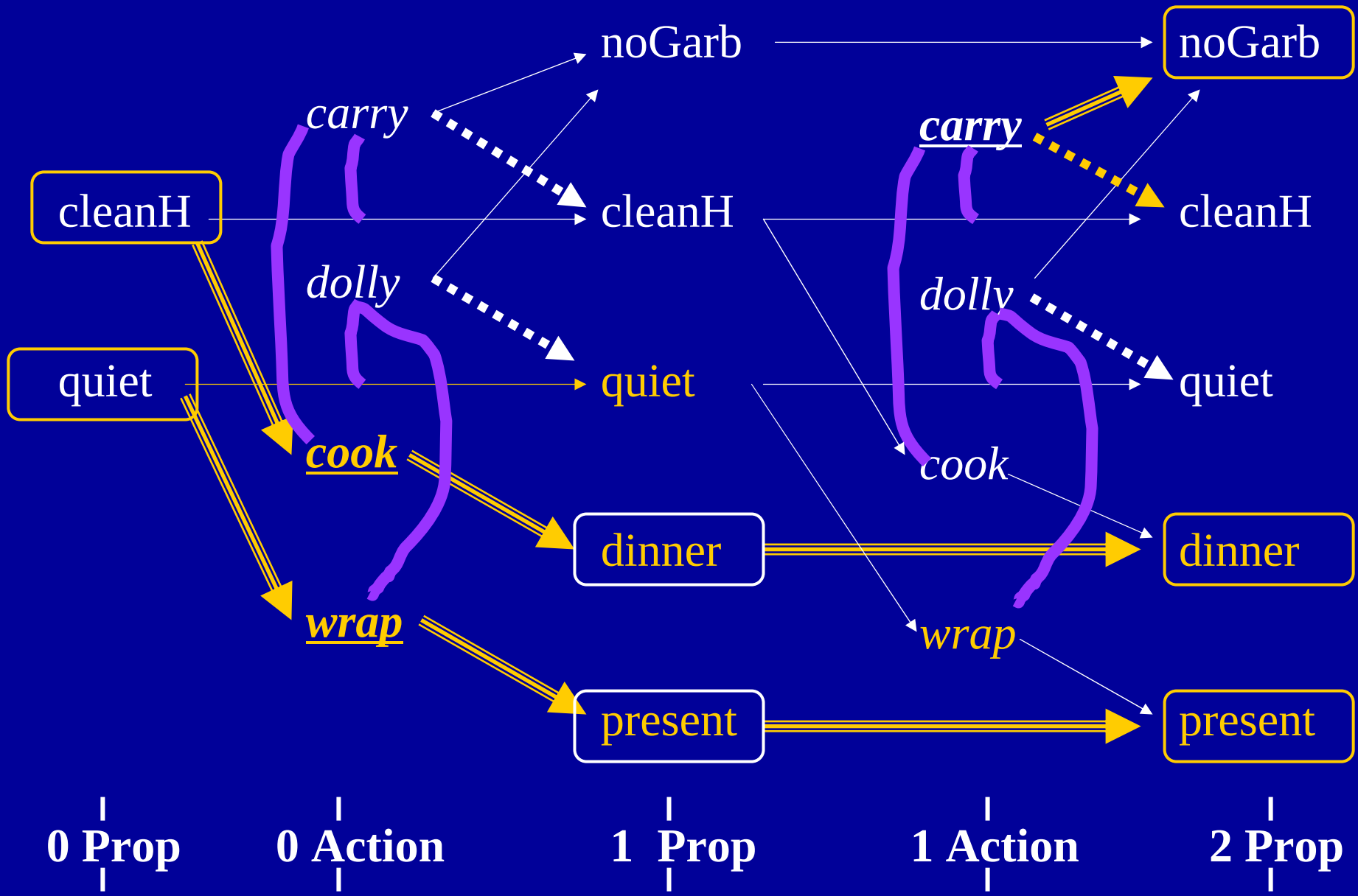
GraphPlan Algorithm

- Phase 1 – Plan Graph Expansion
  - Graph encodes reachability and pairwise consistency of actions and propositions from initial state.
  - Graph includes, as a subset, all plans that are complete and consistent.
- Phase 2 - Solution Extraction
  - Graph treated as a kind of constraint satisfaction problem (CSP).
  - Selects whether or not to perform each action at each time point, by assigning CSP variables and testing consistency.

Outline

- Operator-based Planning
- Graph Plan
  - The Graph Plan Planning Problem
  - Graph Construction
  - Solution Extraction
  - Properties
  - Termination with Failure

Example: Graph and Solution



Graph Properties

A Plan graph

- compactly encodes the space of consistent plans,
- while pruning . . .
  1. partial states and actions at each time i that are not reachable from the initial state.
  2. pairs of propositions and actions that are mutually inconsistent at time i.
  3. plans that cannot reach the goals.

Graph Properties

- Plan graphs are constructed in polynomial time and are of polynomial in size.
- The plan graph does not eliminate all infeasible plans.

➔Plan generation still requires *focused* search.

Constructing the plan graph... (Reachability)

- Initial proposition layer
  - Contains propositions that hold in the initial state.

Example: Initial State, Layer 1

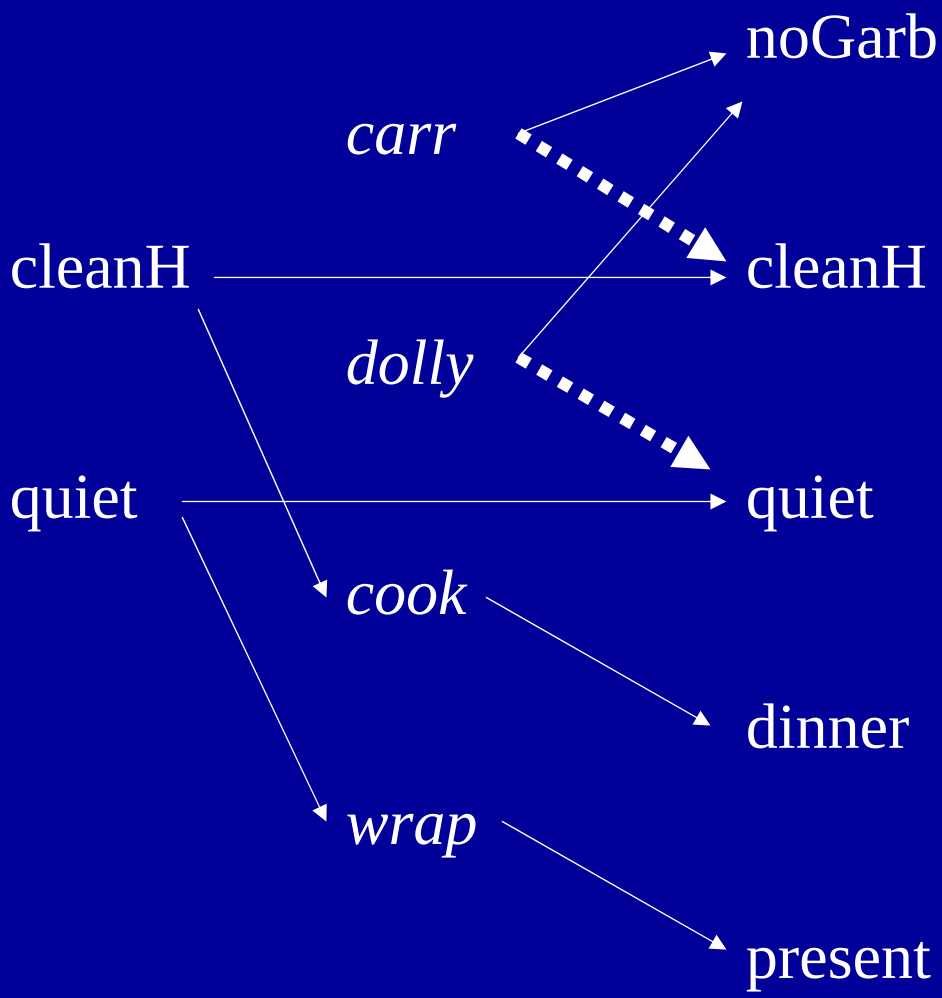
cleanH

quiet

Constructing the plan graph... (Reachability)

- Initial proposition layer
  - Contains propositions in initial state
- Action layer i
  - If all of action's preconditions are consistent in proposition layer i
  - Then add action to layer i
- Proposition layer i+1
  - For each action at layer i
  - Add all its effects at layer i+1

Example: Add Actions and Effects

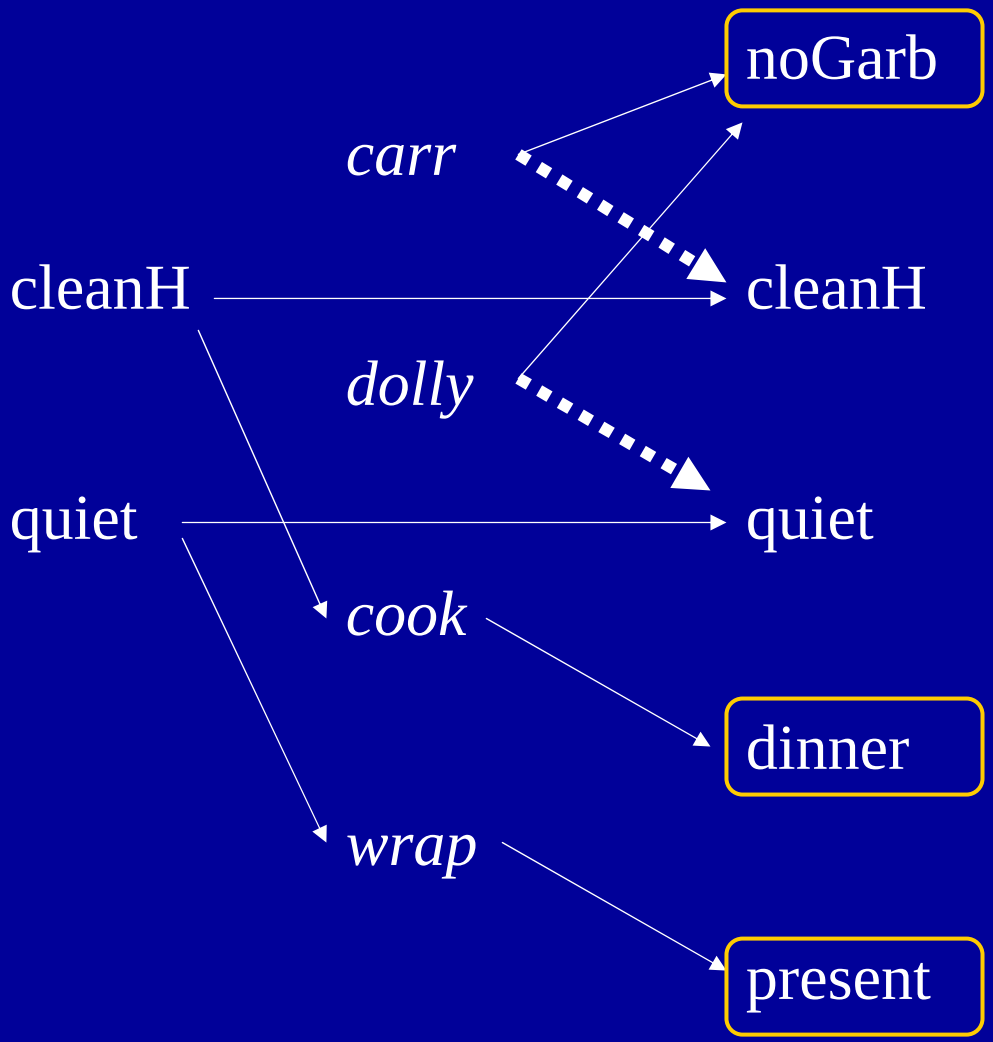


0 Prop **0 Action** **1 Prop** **1 Action** **2 Prop**

Constructing the planning graph...(Reachability)

- Initial proposition layer
  - Write down just the initial conditions
- Action layer i
  - If all action's preconditions appear consistent in proposition layer i
  - Then add action to layer i
- Proposition layer i+1
  - For each action at layer i
  - Add all its effects at layer i+1
- Repeat adding layers until all goal propositions appear

Round 1: Stop at Proposition Layer 1?



Do all goal propositions appear?

Goal: (and (noGarbage)
(dinner)
(present))

0 Prop **0 Action** **1 Prop** **1 Action** **2 Prop**

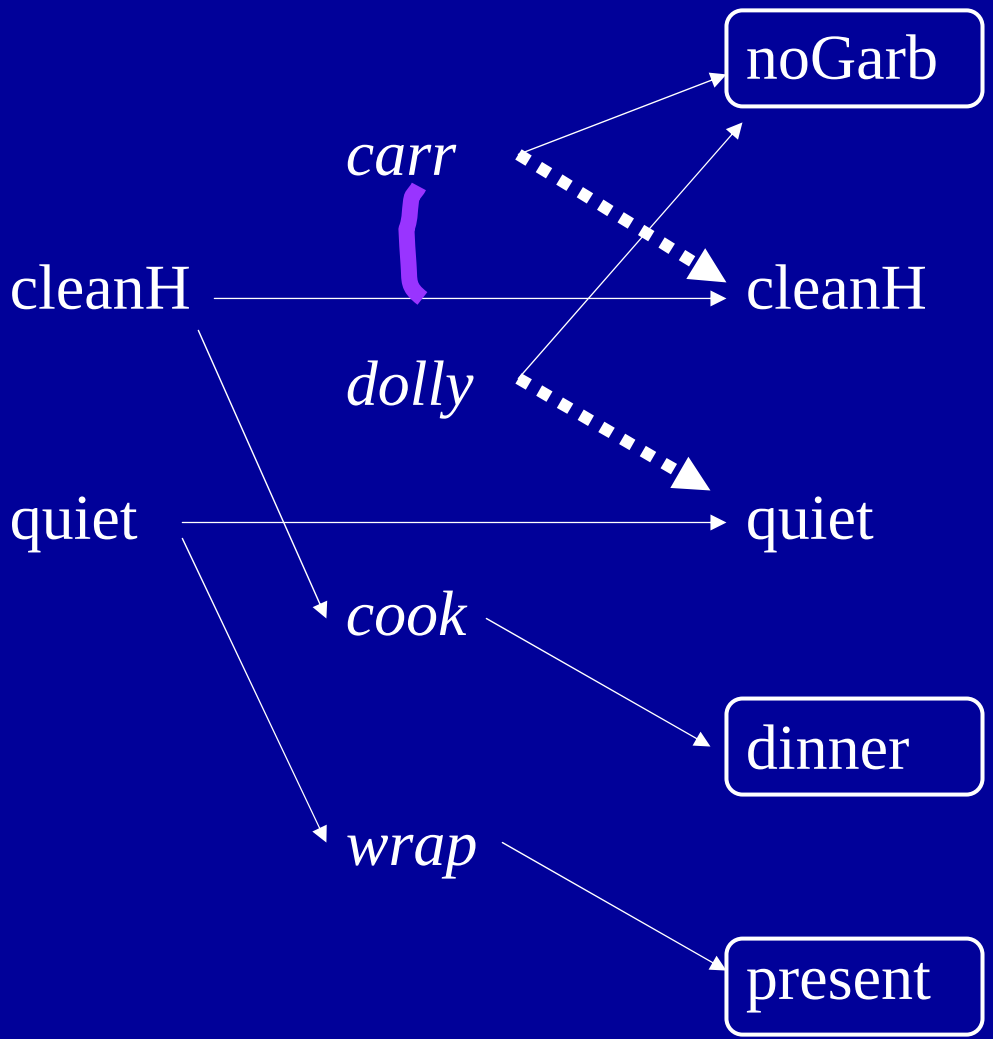
Constructing the plan graph... (Consistency)

- Initial proposition layer
  - Write down just the initial conditions
- Action layer i
  - If action's preconditions appear consistent in i-1 (non-mutex)
  - Then add action to layer i
- Proposition layer i+1
  - For each action at layer i
  - Add all its effects at layer i+1
- Identify mutual exclusions
  - Actions in layer i
  - Propositions in layer i + 1
- Repeat until all goal propositions appear non-mutex

Mutual Exclusion: Actions

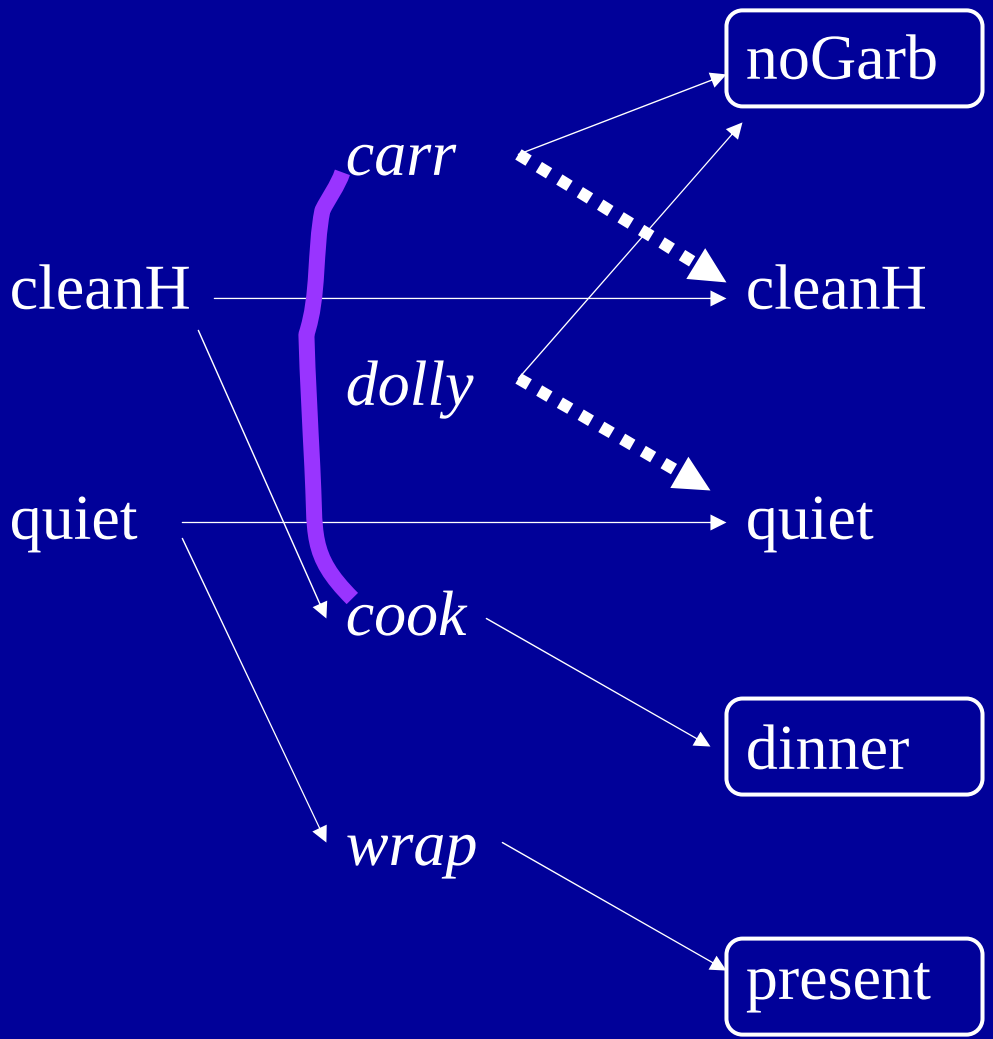
- Actions A,B are **mutually *exclusive at level i*** if no valid plan could possibly contain both at i:
  - They have inconsistent effects.
    - A deletes B's effects,
  - Effects interfere with preconditions.
    - A deletes B's preconditions, or
    - Vice versa or
  - They compete for needs.
    - A and B have inconsistent preconditions

Mutual exclusion: Actions



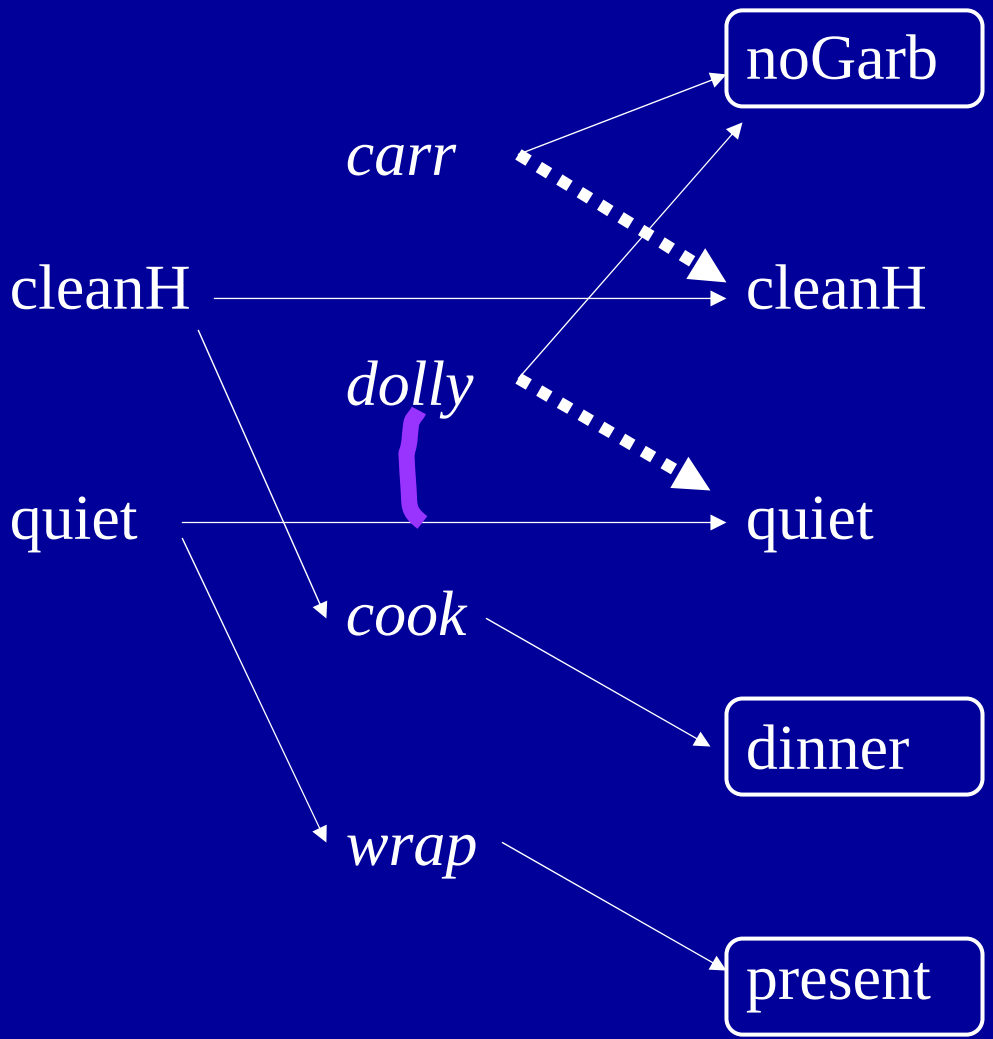
0 Prop **0 Action** **1 Prop** **1 Action** **2 Prop**

Mutual exclusion: Actions



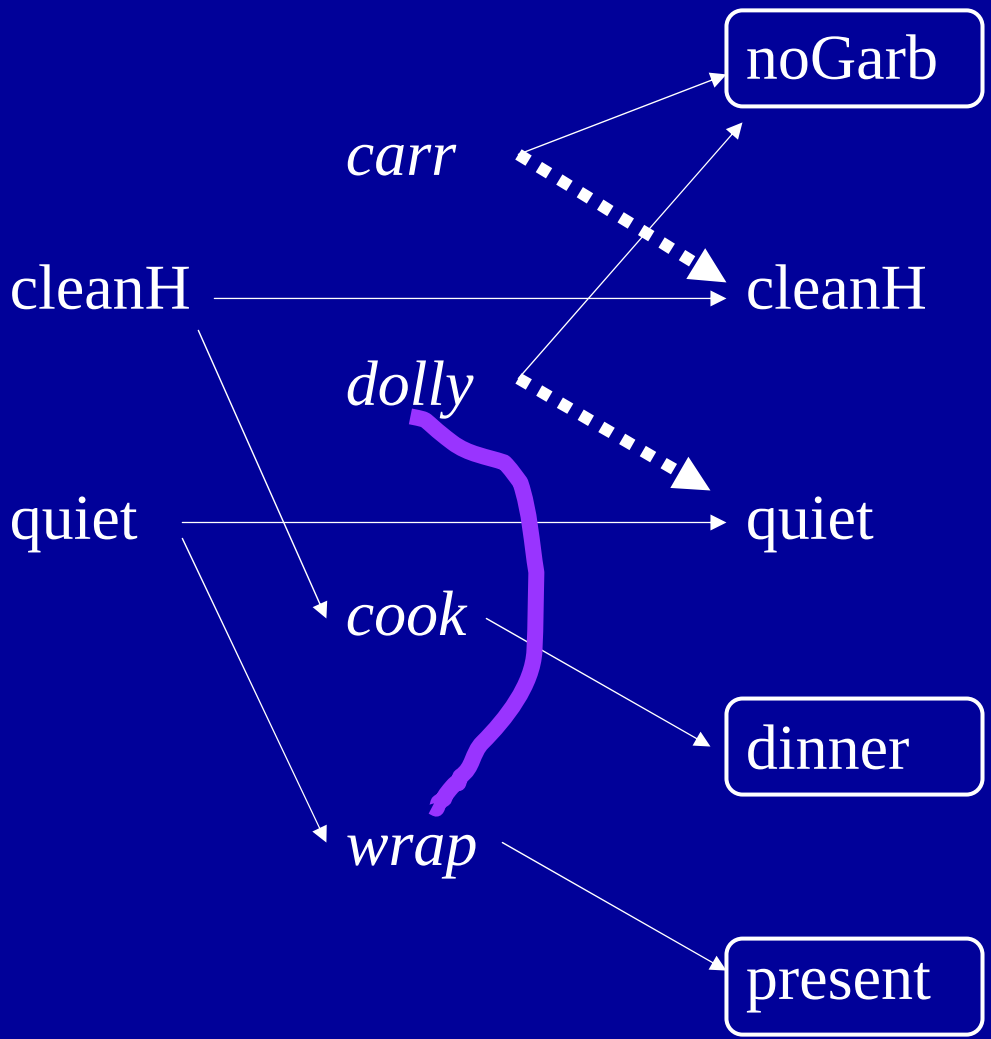
0 Prop **0 Action** **1 Prop** **1 Action** **2 Prop**

Mutual exclusion: Actions



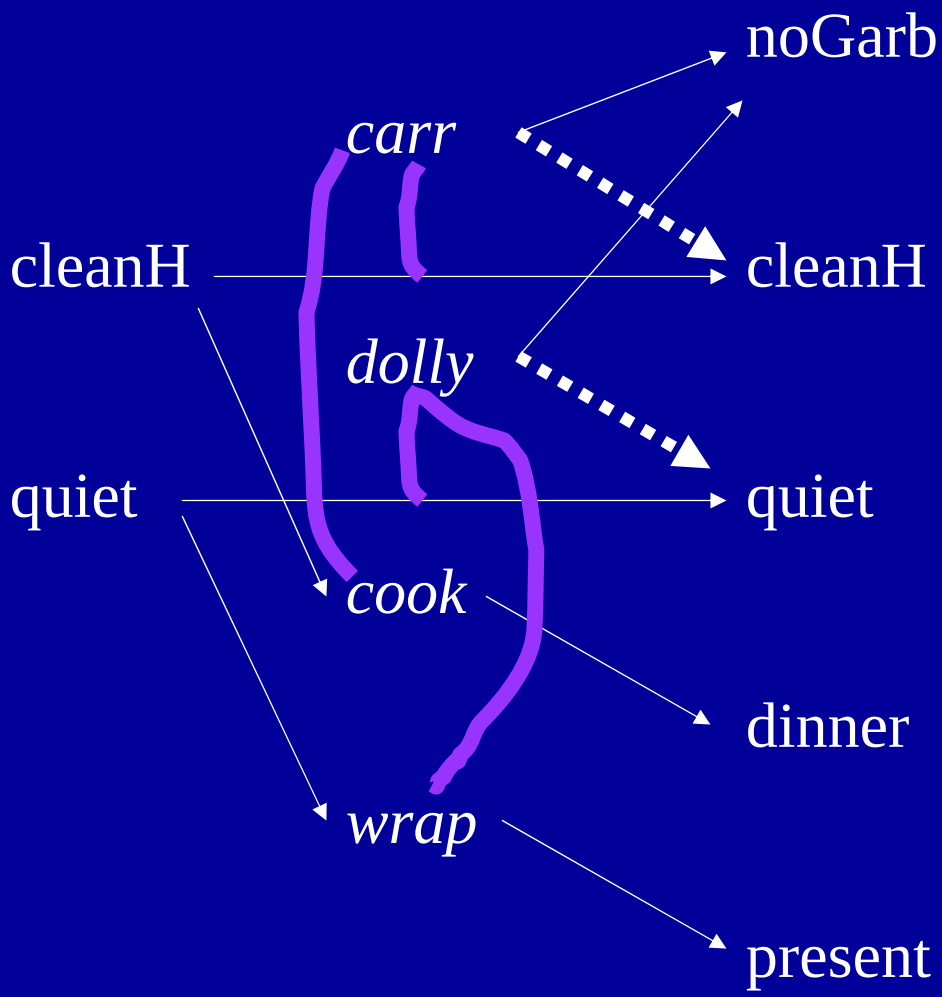
0 Prop **0 Action** **1 Prop** **1 Action** **2 Prop**

Mutual exclusion: Actions



0 Prop **0 Action** **1 Prop** **1 Action** **2 Prop**

Layer 1: complete action mutexs



0 Prop **0 Action** **1 Prop** **1 Action** **2 Prop**

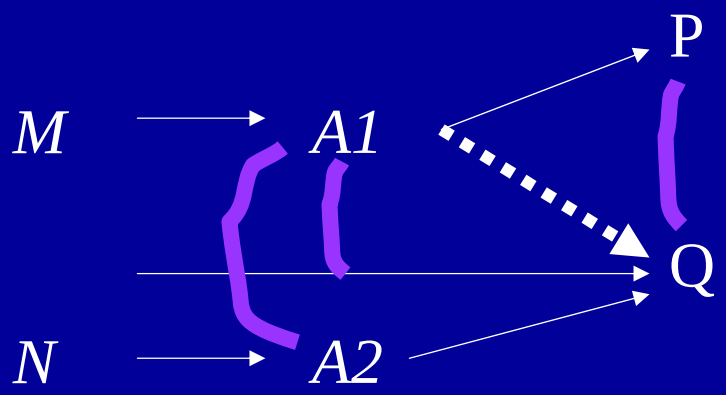
Mutual Exclusion: Actions

- Actions A,B are **mutually *exclusive at level i*** if no valid plan could possibly contain both at i:
  - They **Interfere**
    - A deletes B's preconditions, or
    - Vice versa
  - They have **inconsistent effects**:
    - A deletes B's effects, or
    - Vice versa
  - They have **competing needs**:
    - A & B have inconsistent preconditions

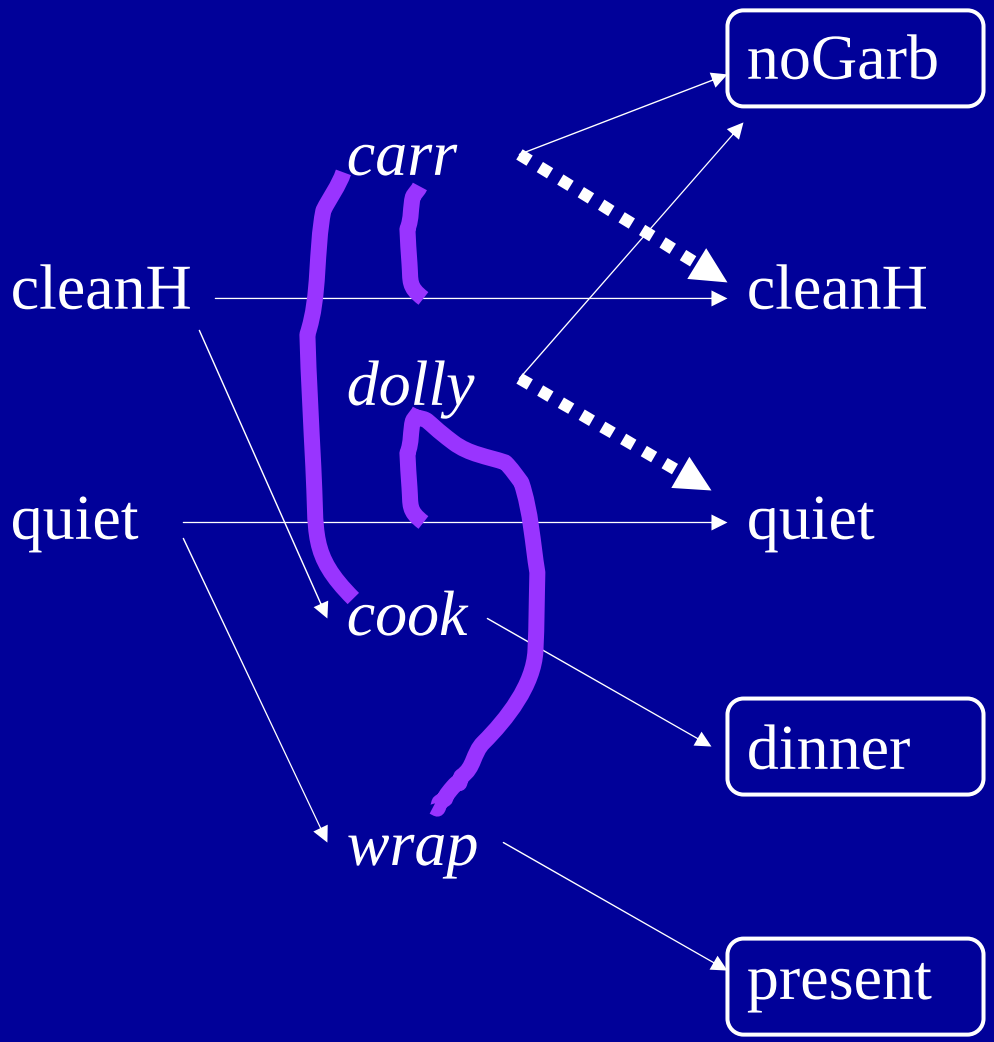
Mutual Exclusion: Proposition Layer

Propositions P,Q are ***inconsistent at i***

- if no valid plan could possibly contain both at i
- If at i, all ways to achieve P exclude all ways to achieve Q



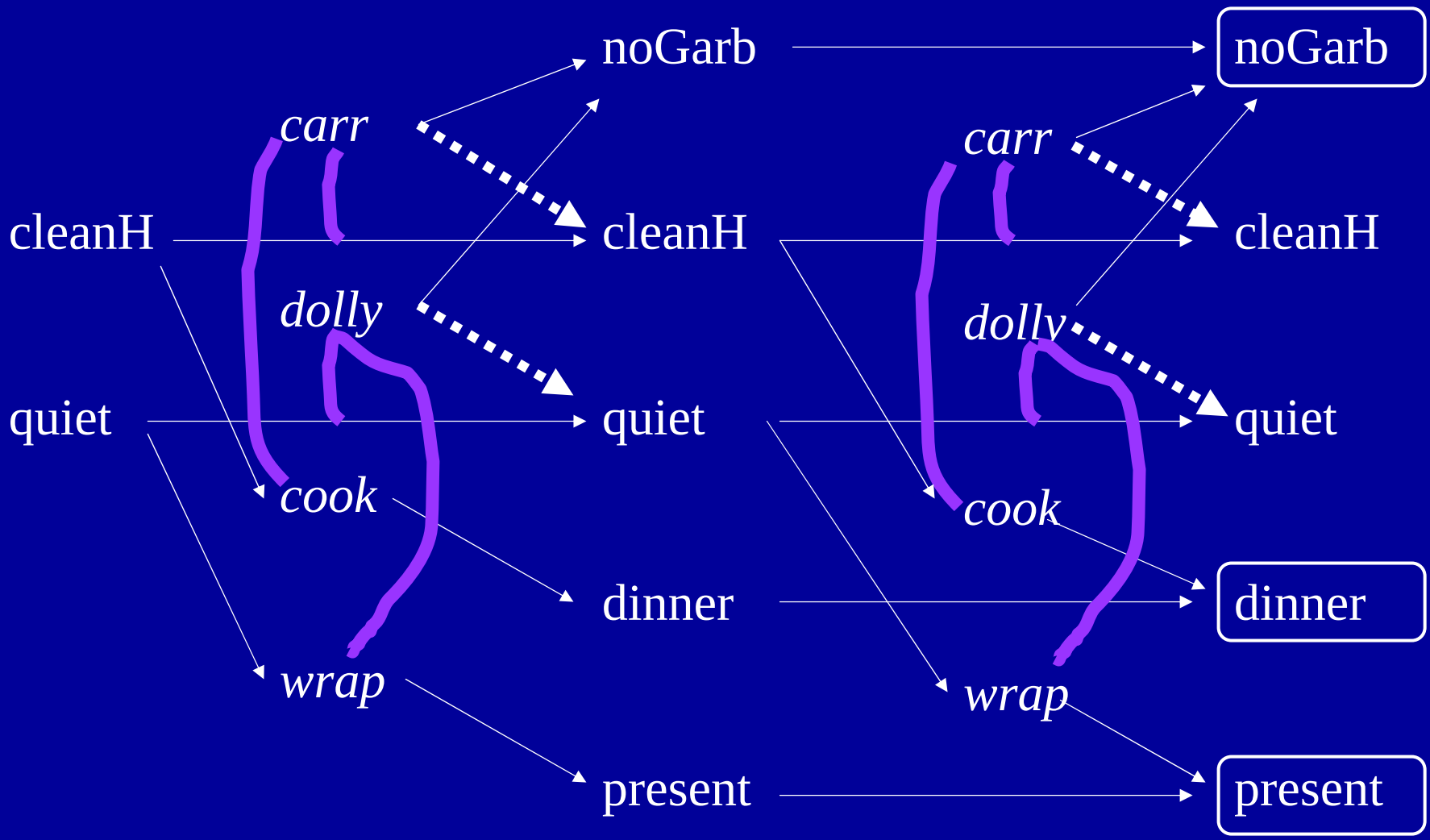
Layer 1: Add Proposition Mutexs



Do all goal propositions appear non-mutex?

0 Prop **0 Action** **1 Prop** **1 Action** **2 Prop**

Round 2: Extending The Plan Graph



0 Prop **0 Action** **1 Prop** **1 Action** **2 Prop**

Outline

- Operator-based Planning
- Graph Plan
  - The Graph Plan Planning Problem
  - Graph Construction
  - Solution Extraction
  - Properties
  - Termination with Failure

Graphplan

- Create plan graph level 1 from initial state
- Loop
  1. If goal $\subseteq$ propositions of the highest level (nonmutex)
  2. Then search graph for solution
     - If solution found, then return and terminate
  3. Extend graph one more level

A kind of double search: forward direction checks necessary (but insufficient) conditions for a solution, ... Backward search verifies...

2. Search for a Solution

Recursively find consistent actions achieving all goals at time t, then time t-1, . . . :

- Find action to achieve each goal G at time t
  - For each action A making G true at t
    - If A isn't mutex with previously chosen action at t, Then select it
  - Finally,
    - If no action of G works,
    - Then backtrack on previous G.
- Finally
  - If action found for each goal at time t,
  - Then recurse on preconditions of actions selected, t-1,
  - Else backtrack to next solution at t+1.

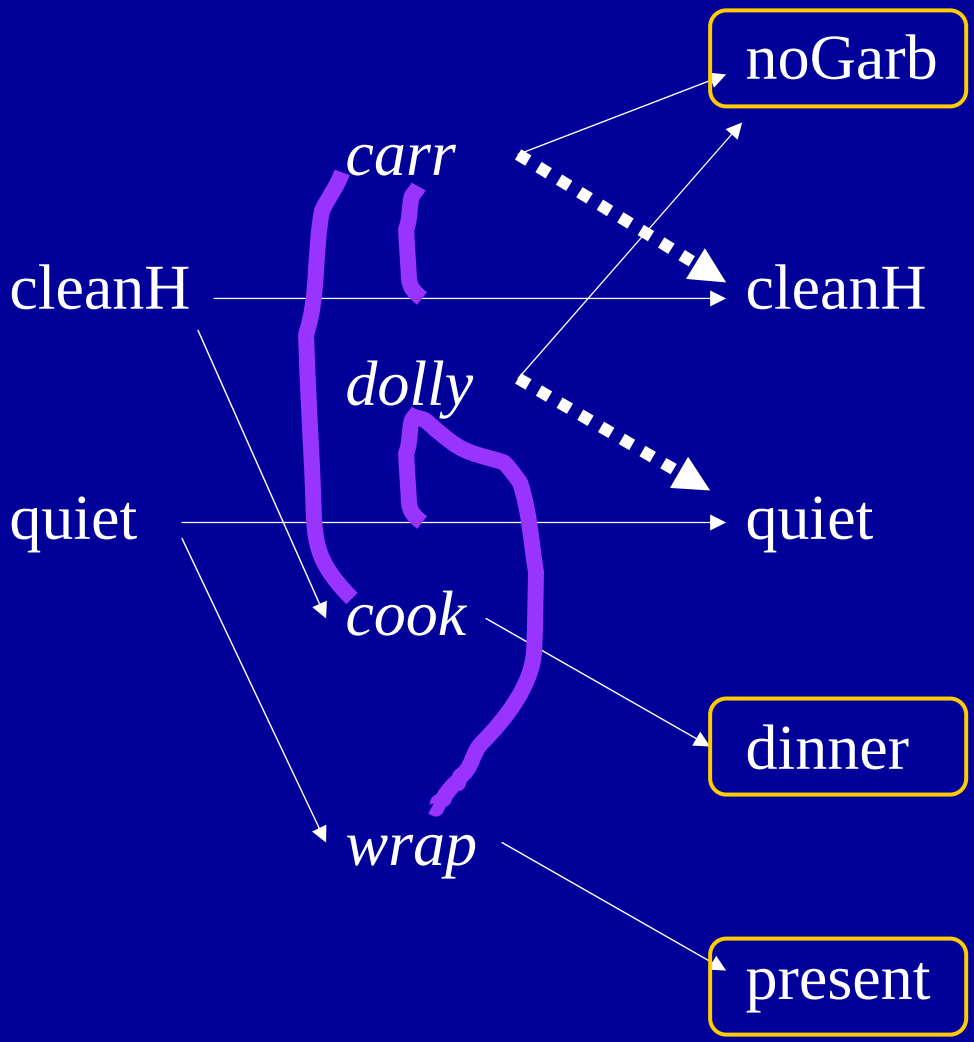
Searching for a Solution

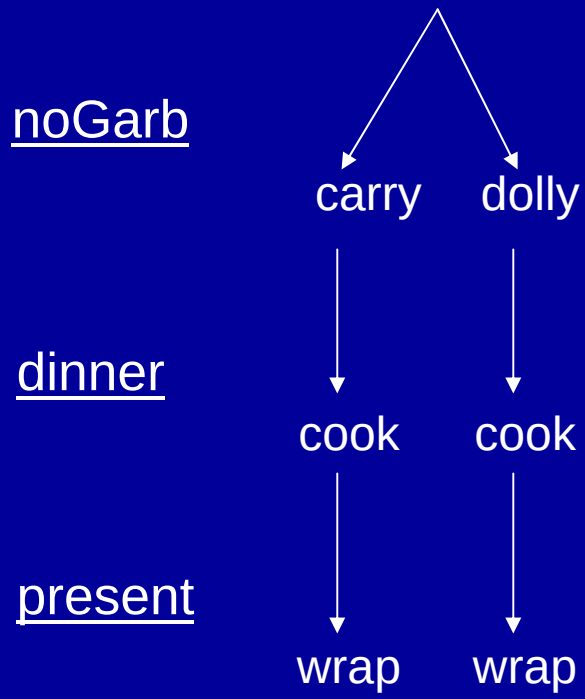
Recursively find consistent actions achieving all goals at time t, then time t-1, . . . :

- Select actions at t-1 to achieve each goal G at t, by solving CSP_t:
  - Variables: One for each goal G_i
  - Domain: For variable G_i, all actions in layer t-1 that add G_i.
  - Constraints: Action mutex of layer t-1
- Finally
  - If solution to CSP found (action found for each goal at time t),
  - Then recurse on preconditions of actions selected for layer t-1,
  - Else backtrack to next solution at t+1.

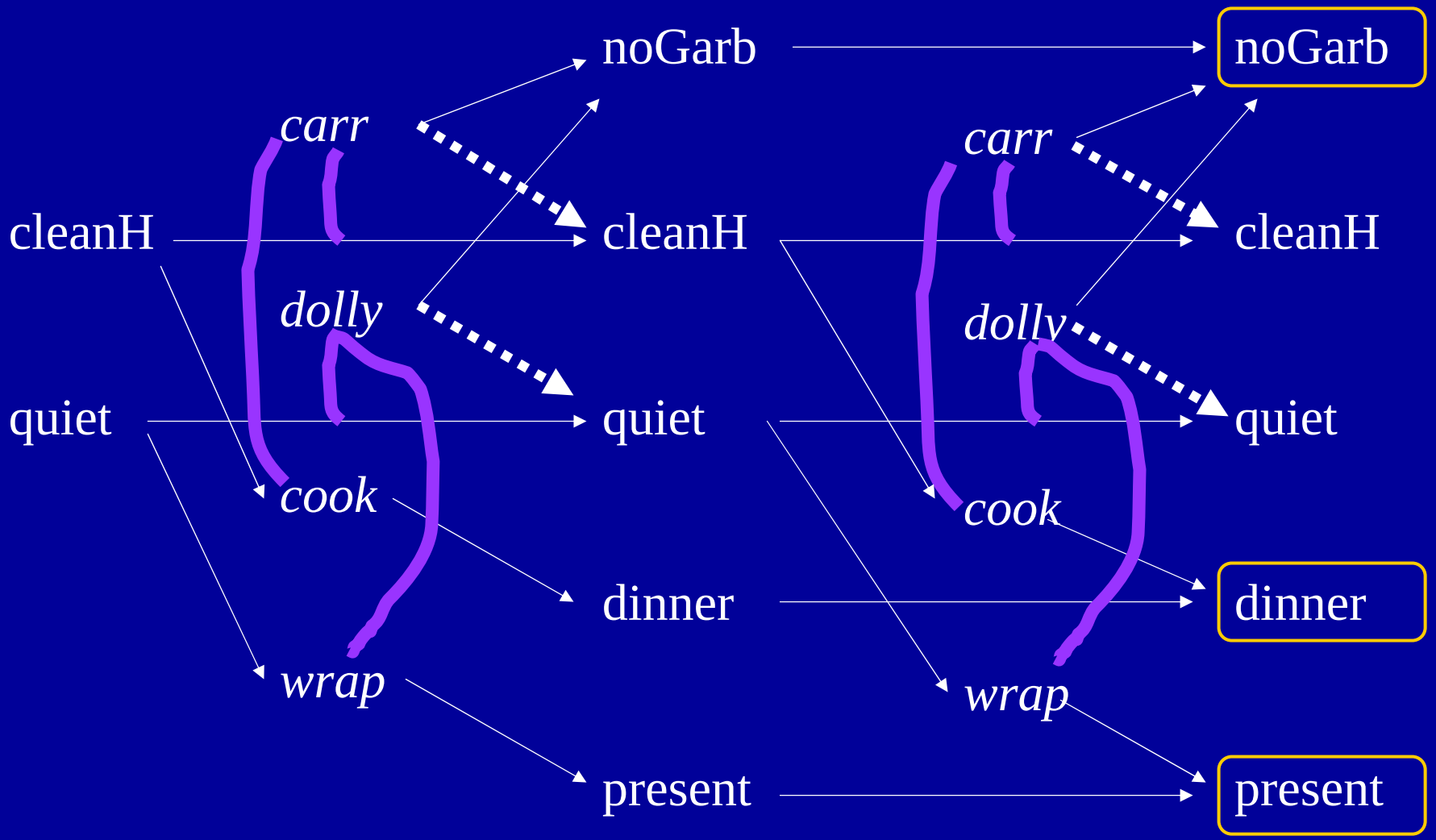
- No-ops are always favored.
  - To guarantee that the plan will not contain redundant plan steps.

Search Action Layer 0



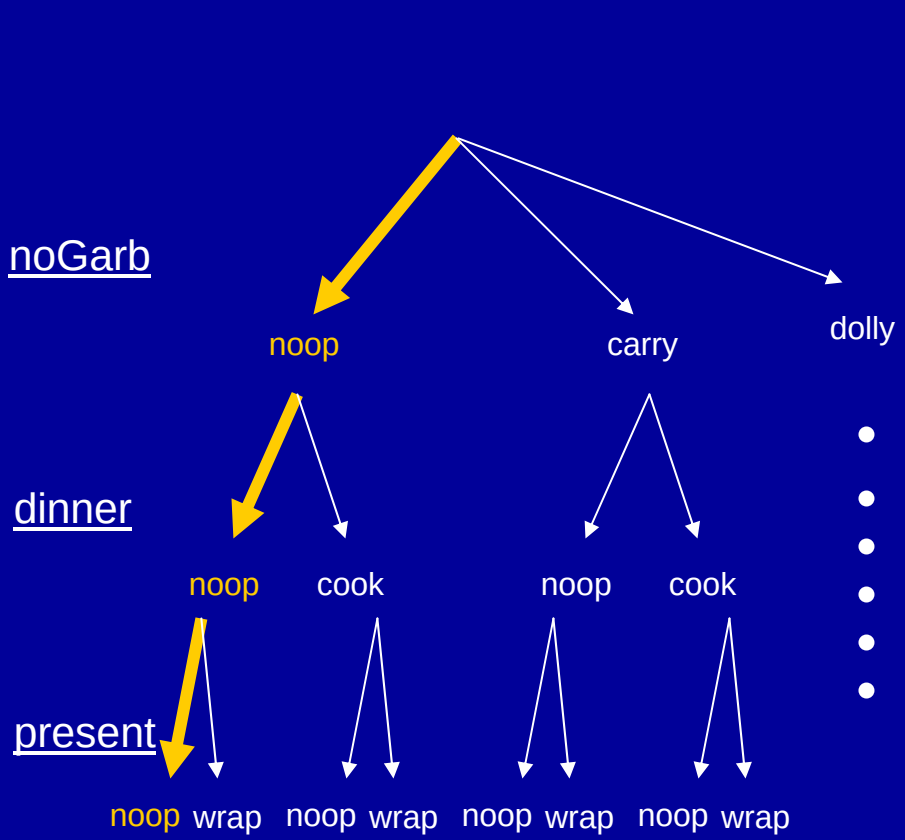


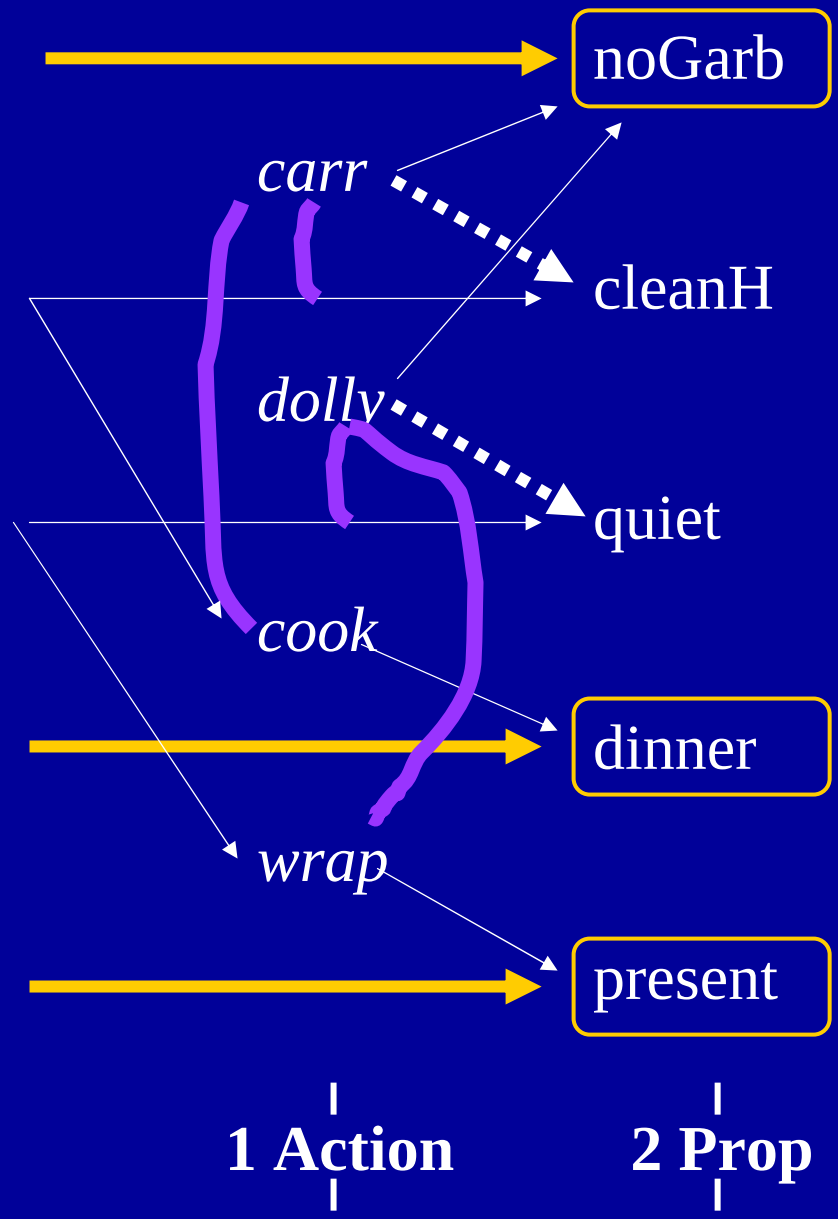
Extend & Search Action Layer 1



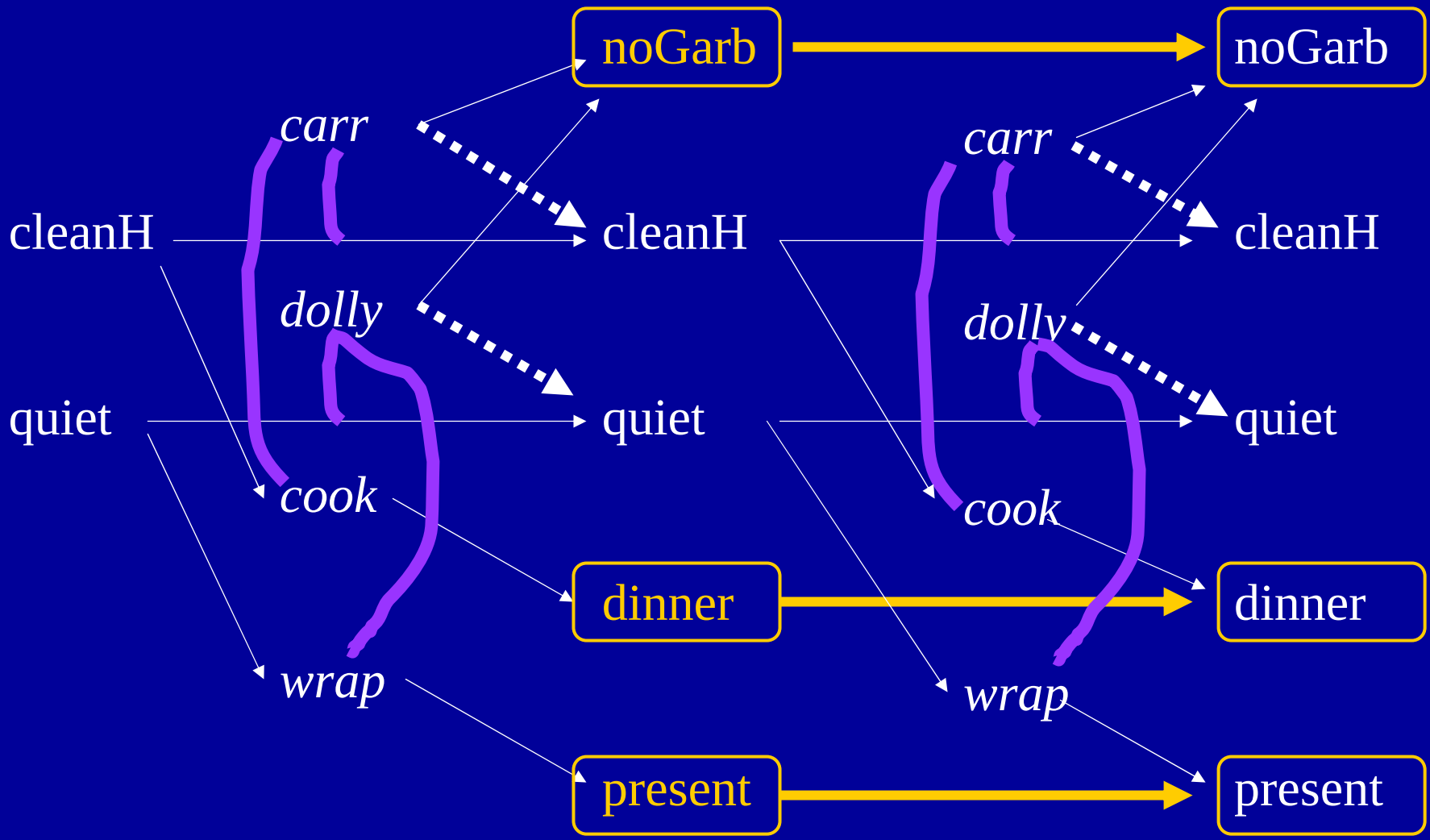
0 Prop **0 Action** **1 Prop** **1 Action** **2 Prop**

Search Action Layer 1

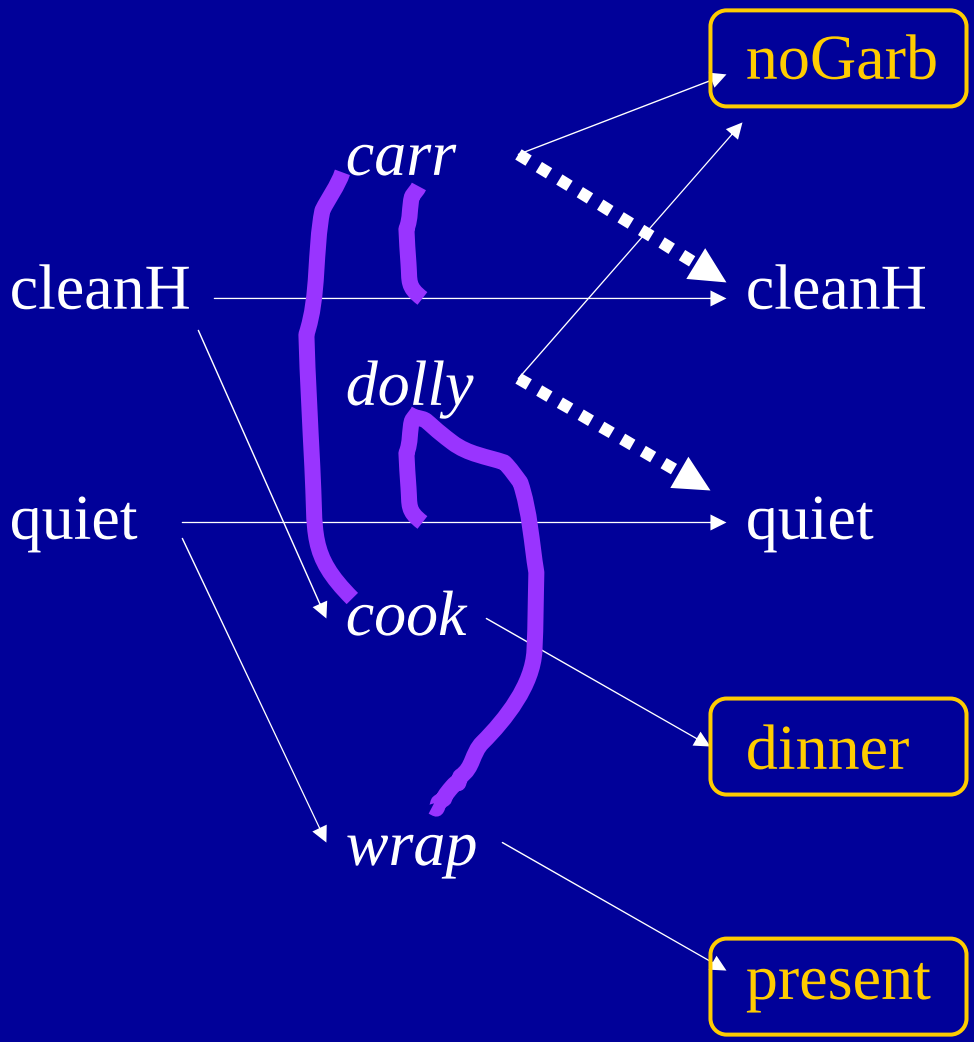


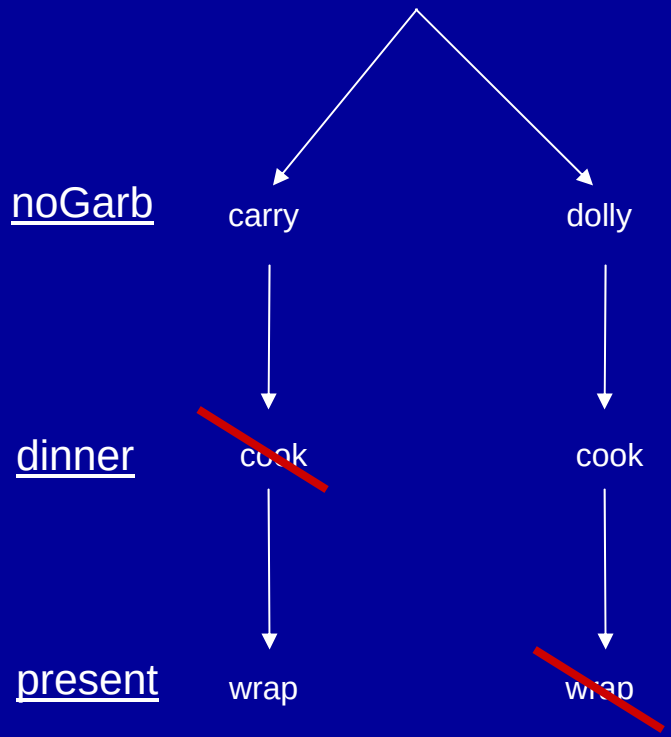


Search Action Layer 0



Search Action Layer 0

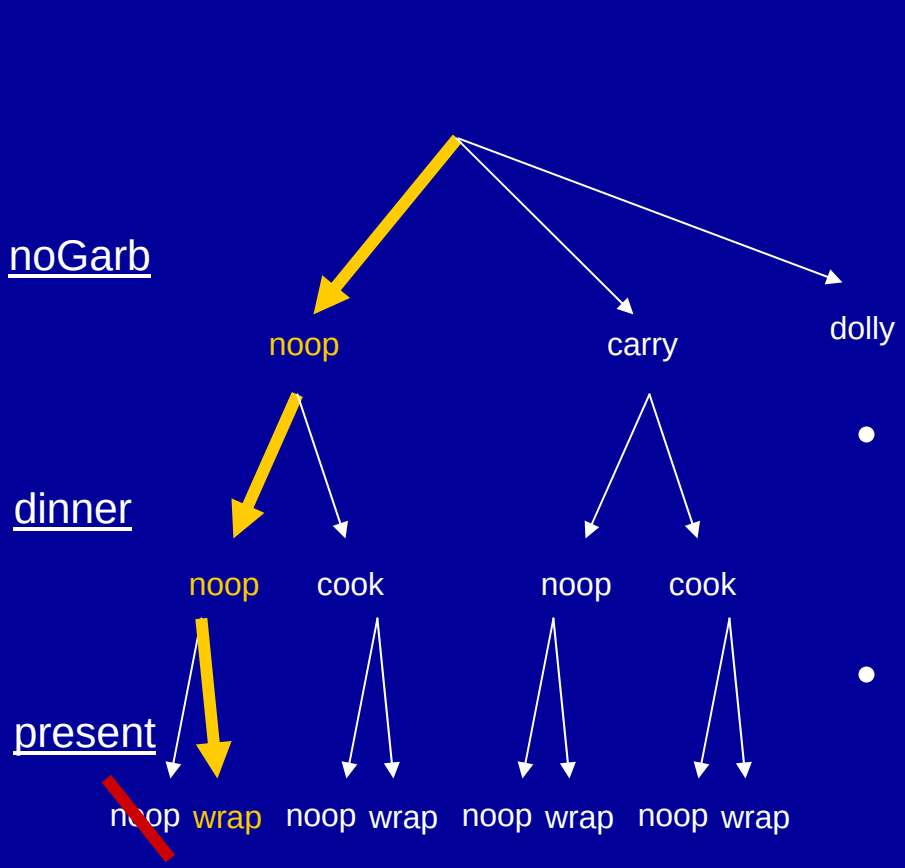


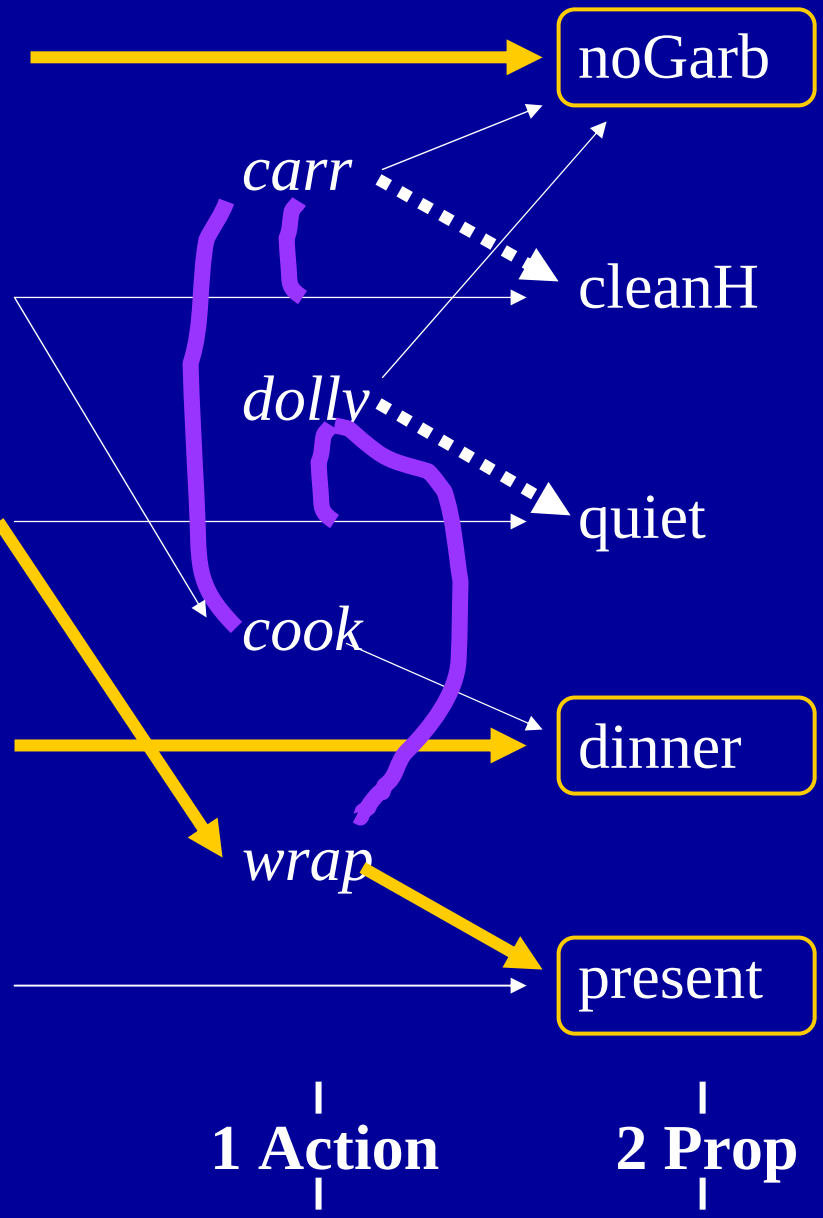


Backtrack!

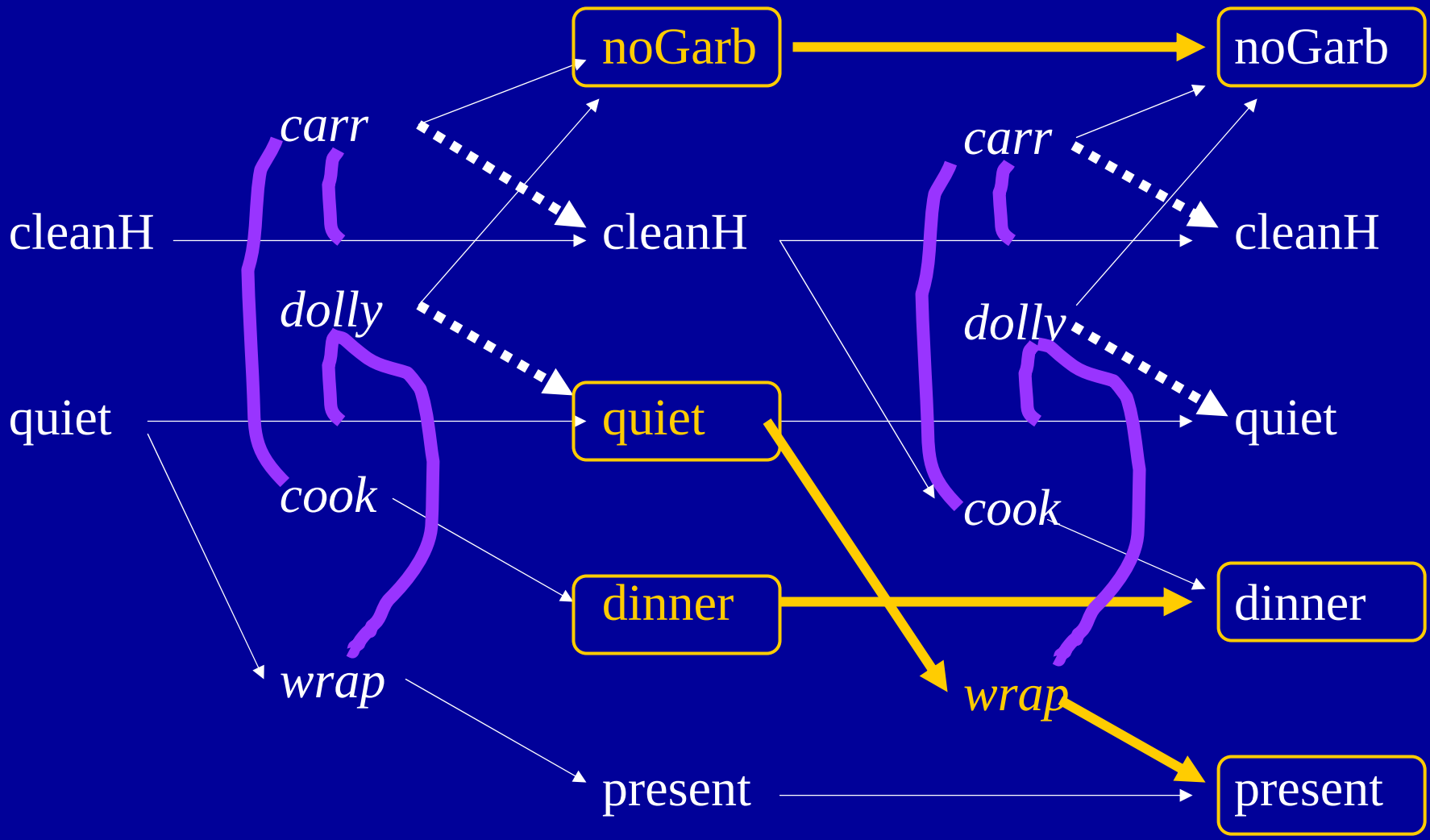
0 Prop **0 Action** **1 Prop**

Search Action Layer 1 Again!

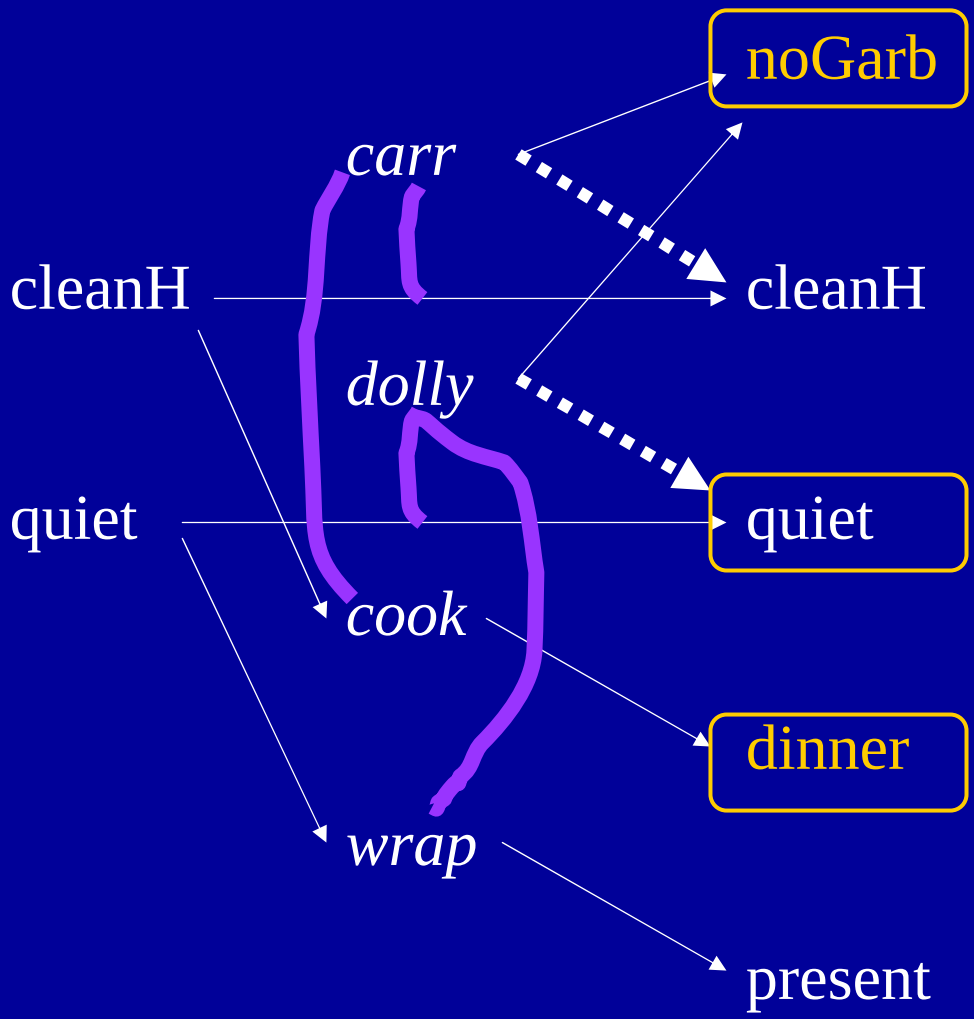


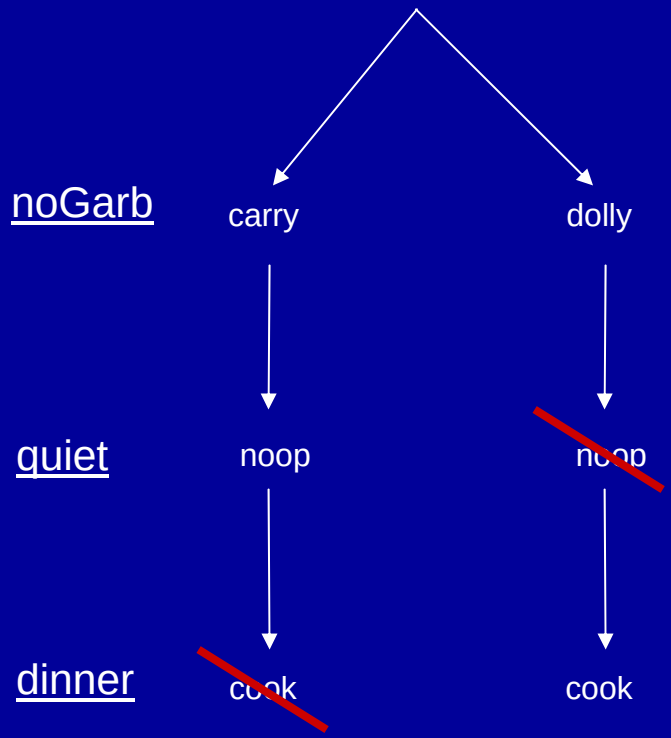


Search Action Layer 0



Search Action Layer 0

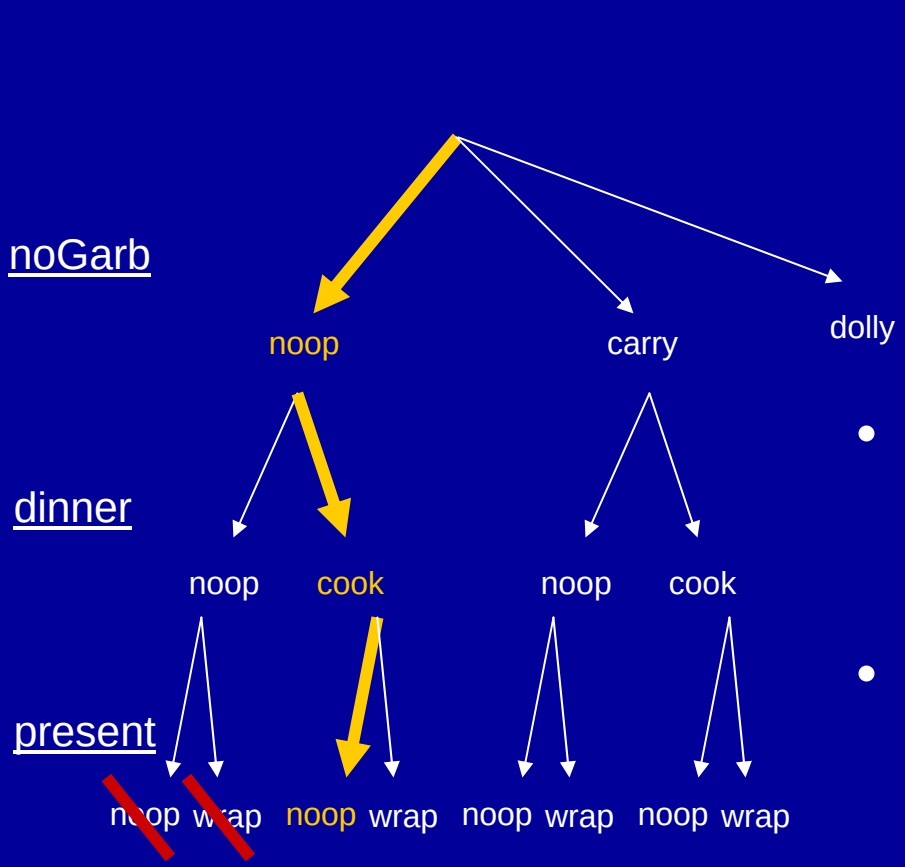


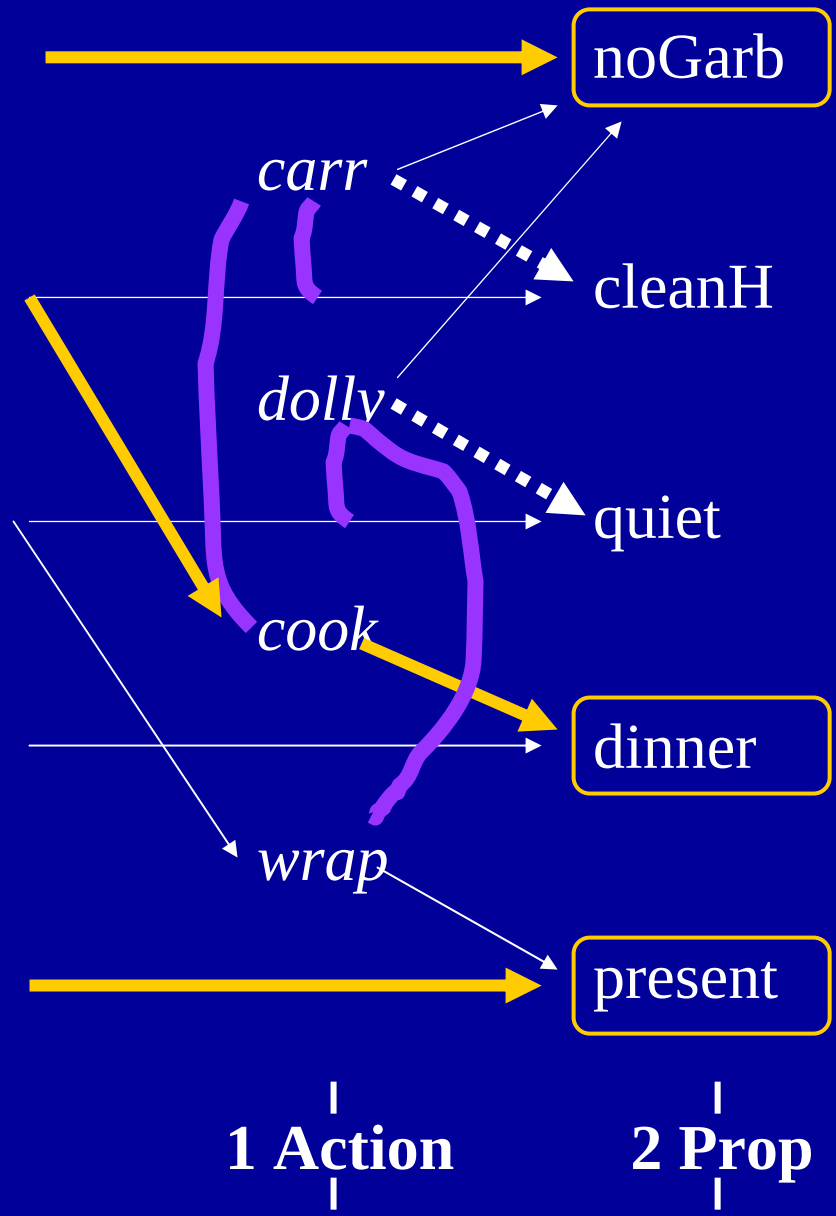


Backtrack!

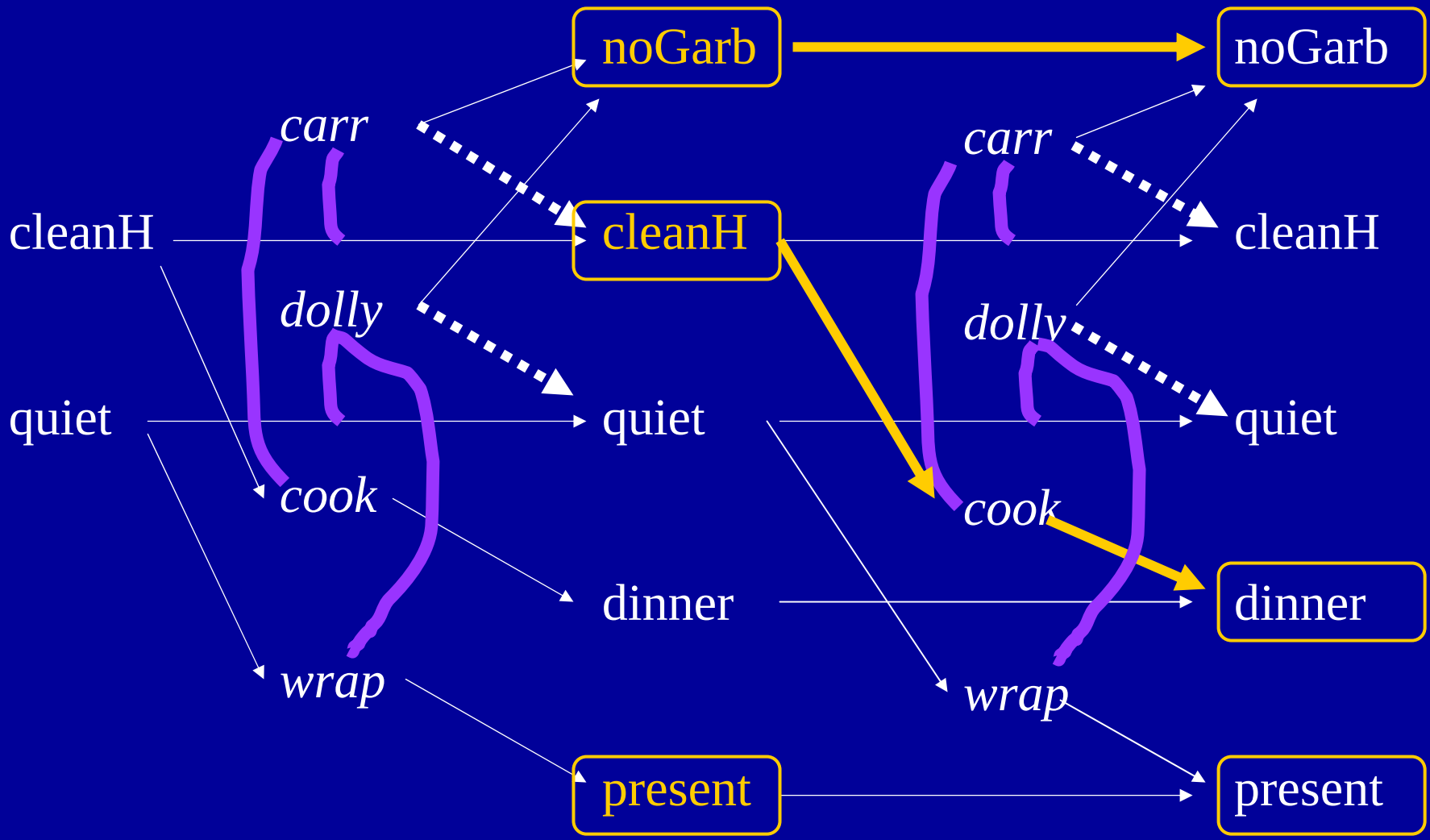
0 Prop **0 Action** **1 Prop**

Search Action Layer 1 Again!

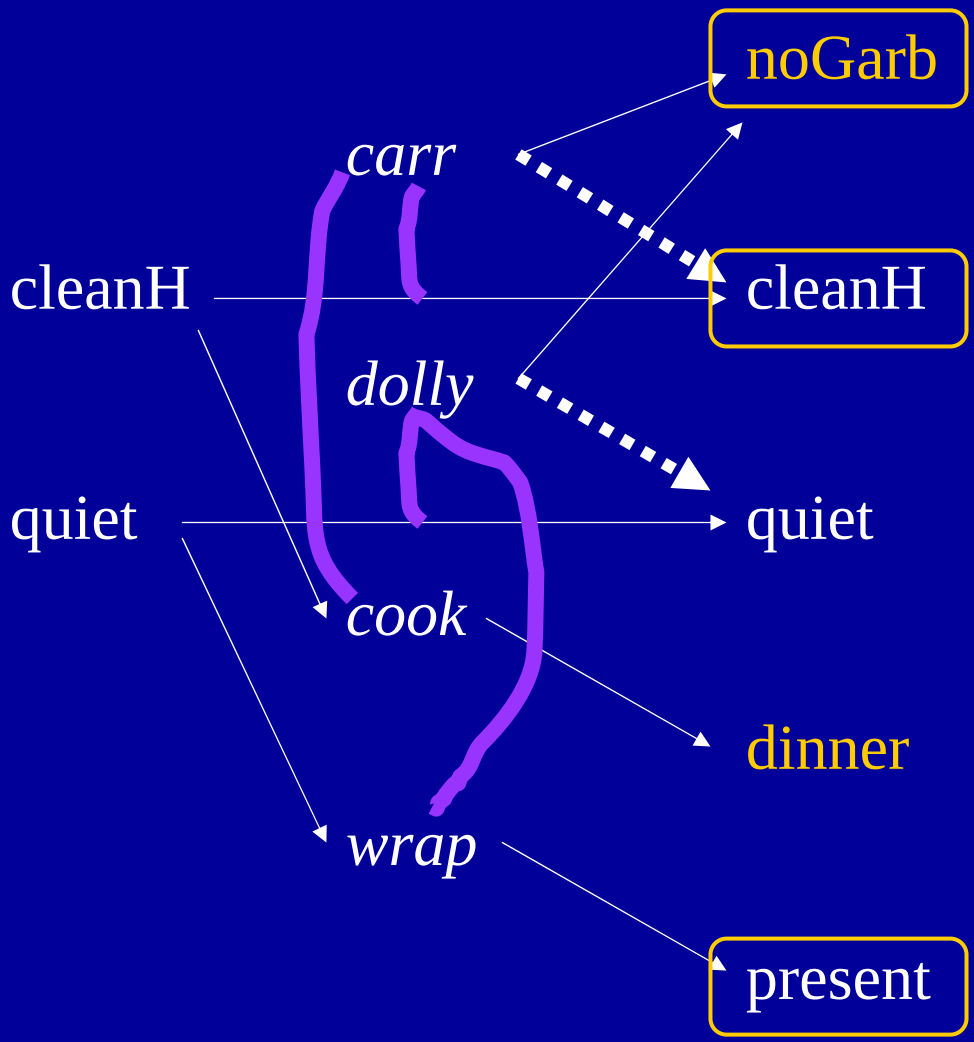


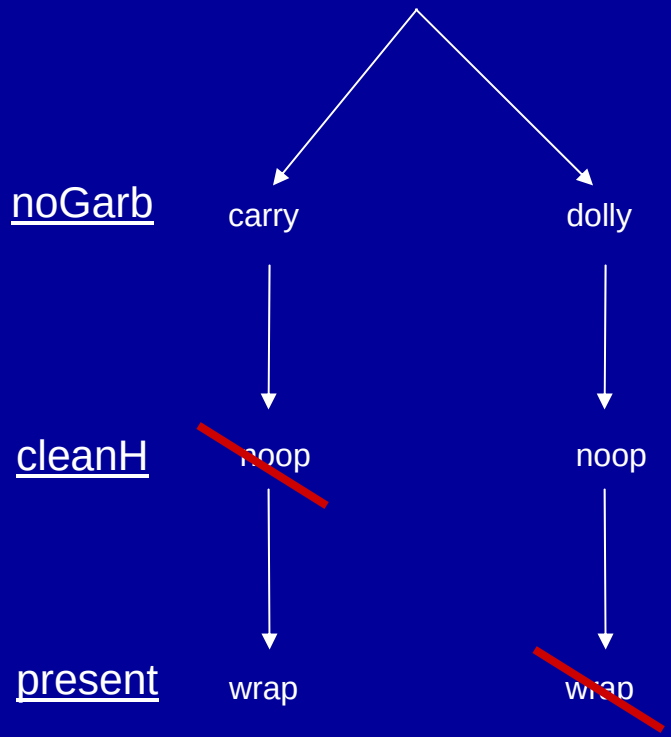


Search Action Layer 0



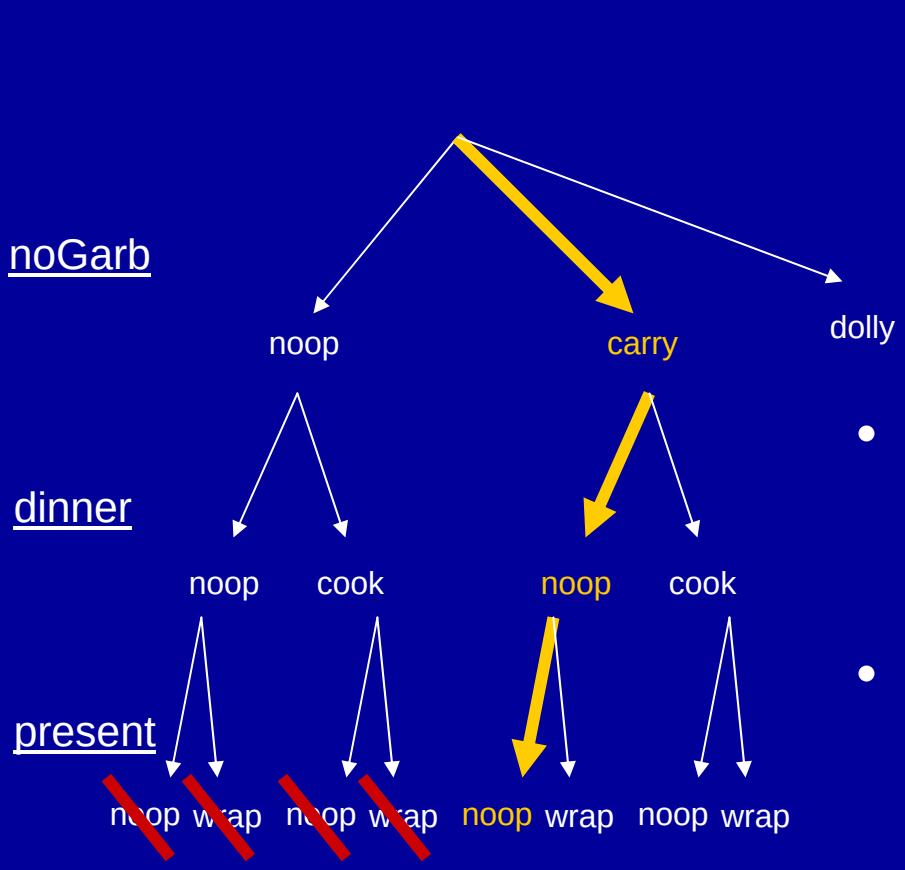
Search Action Layer 0

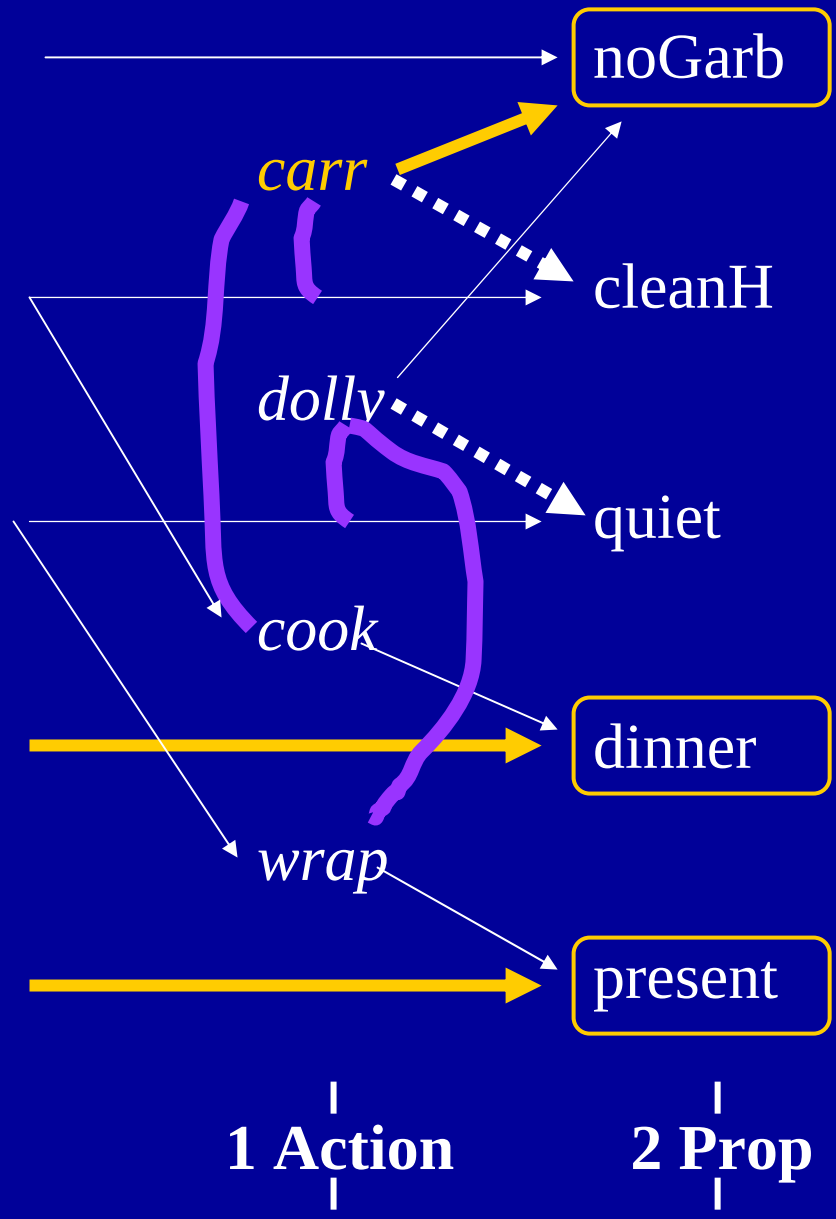




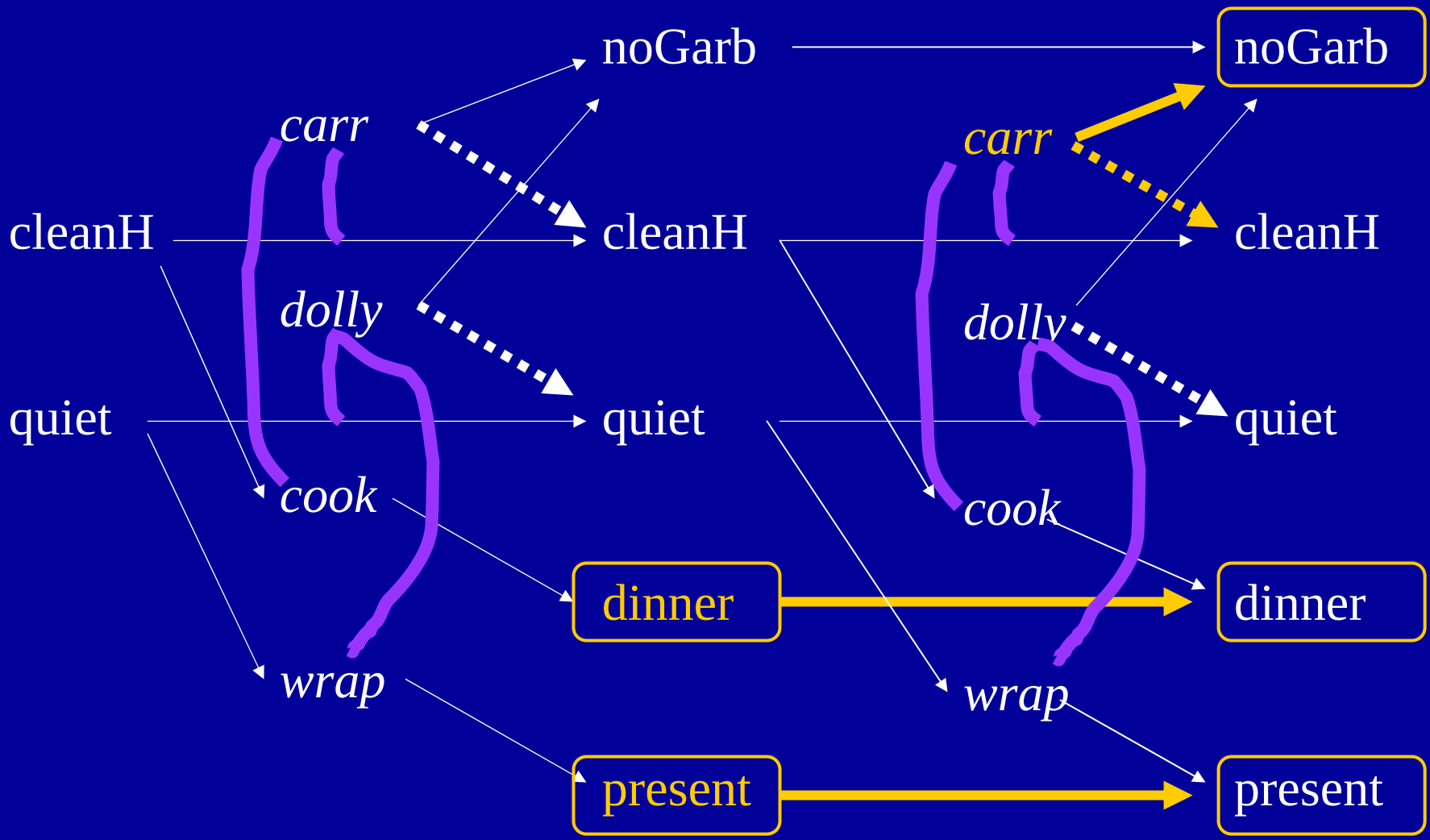
Backtrack!

Search Action Layer 1 Again!

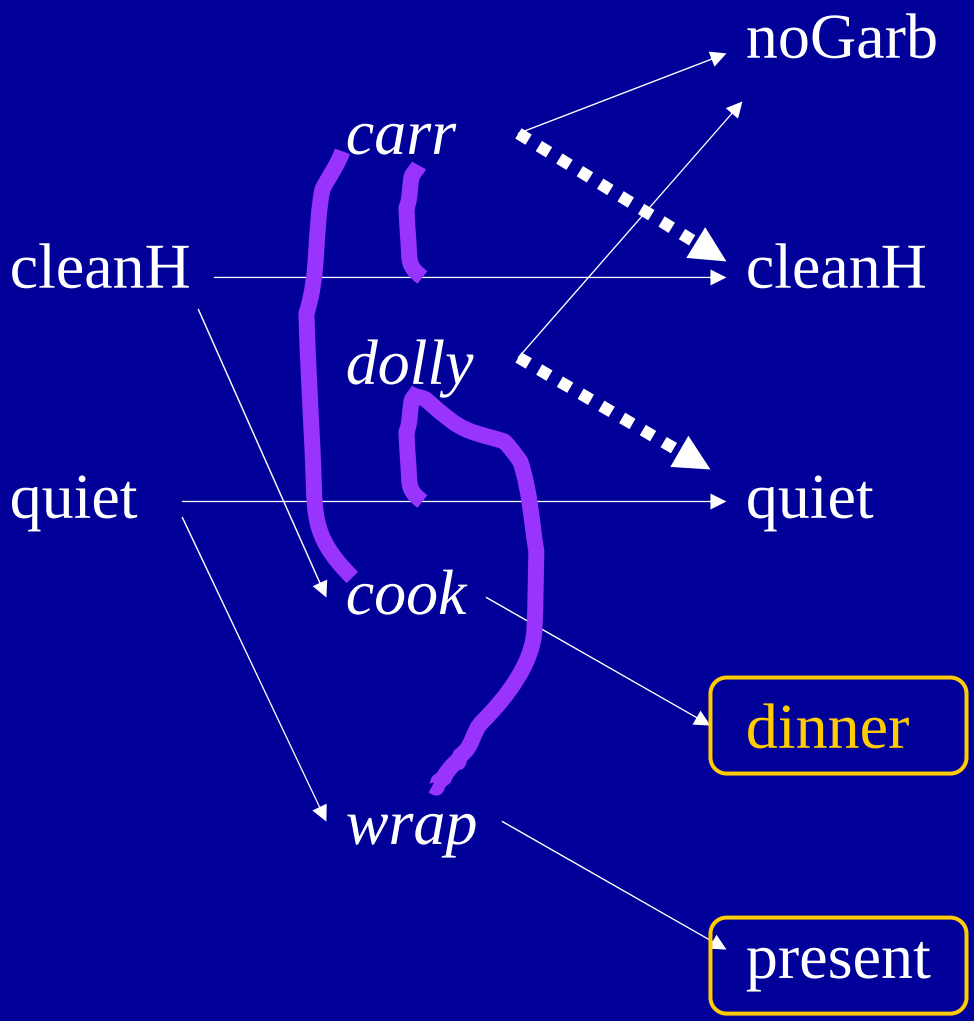


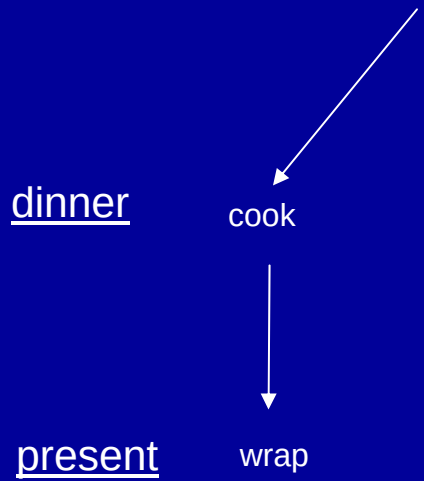


Search Action Layer 0



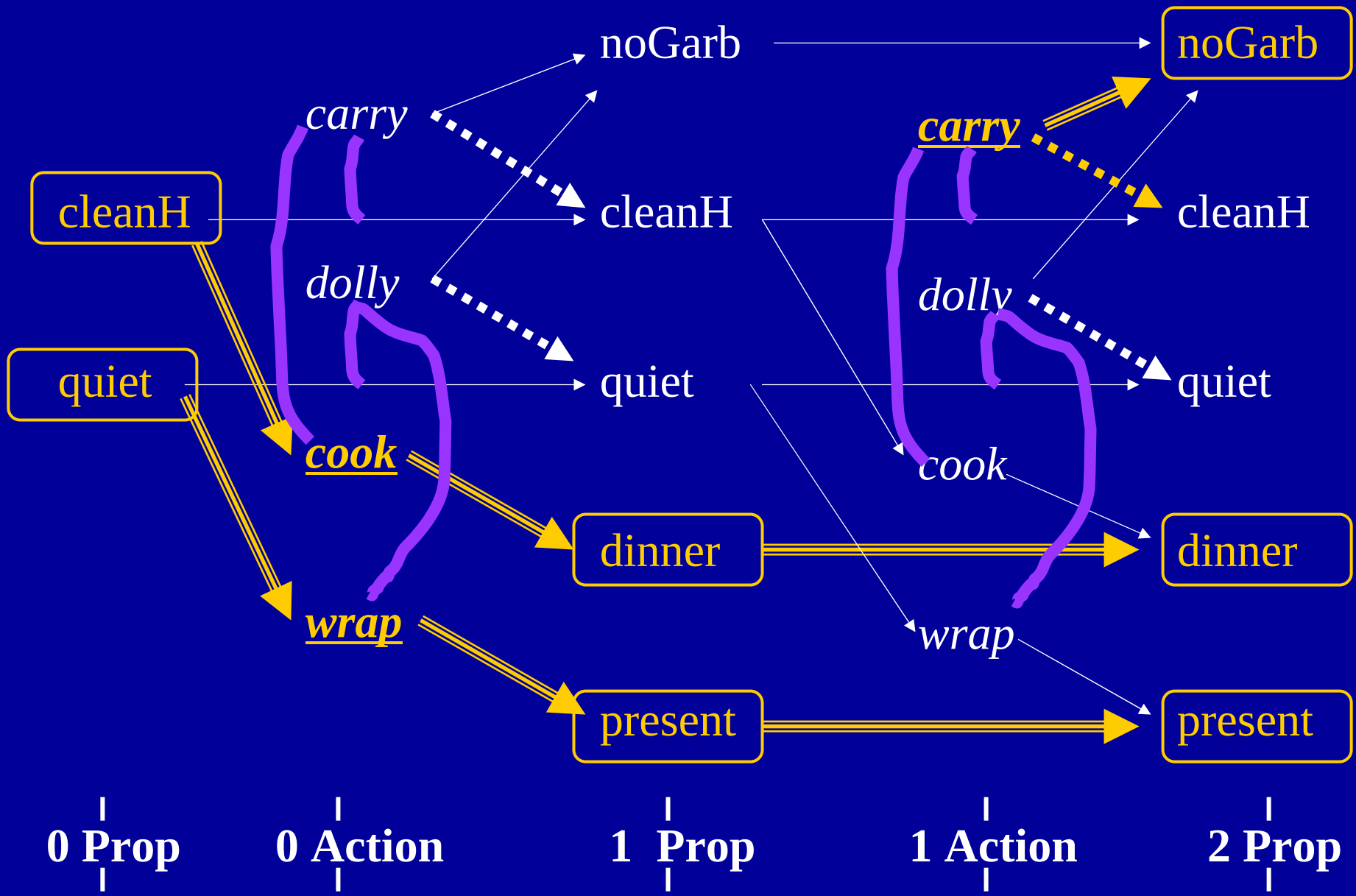
Search Action Layer 0





Consistent!

Solution: Cook & Wrap, then Carry

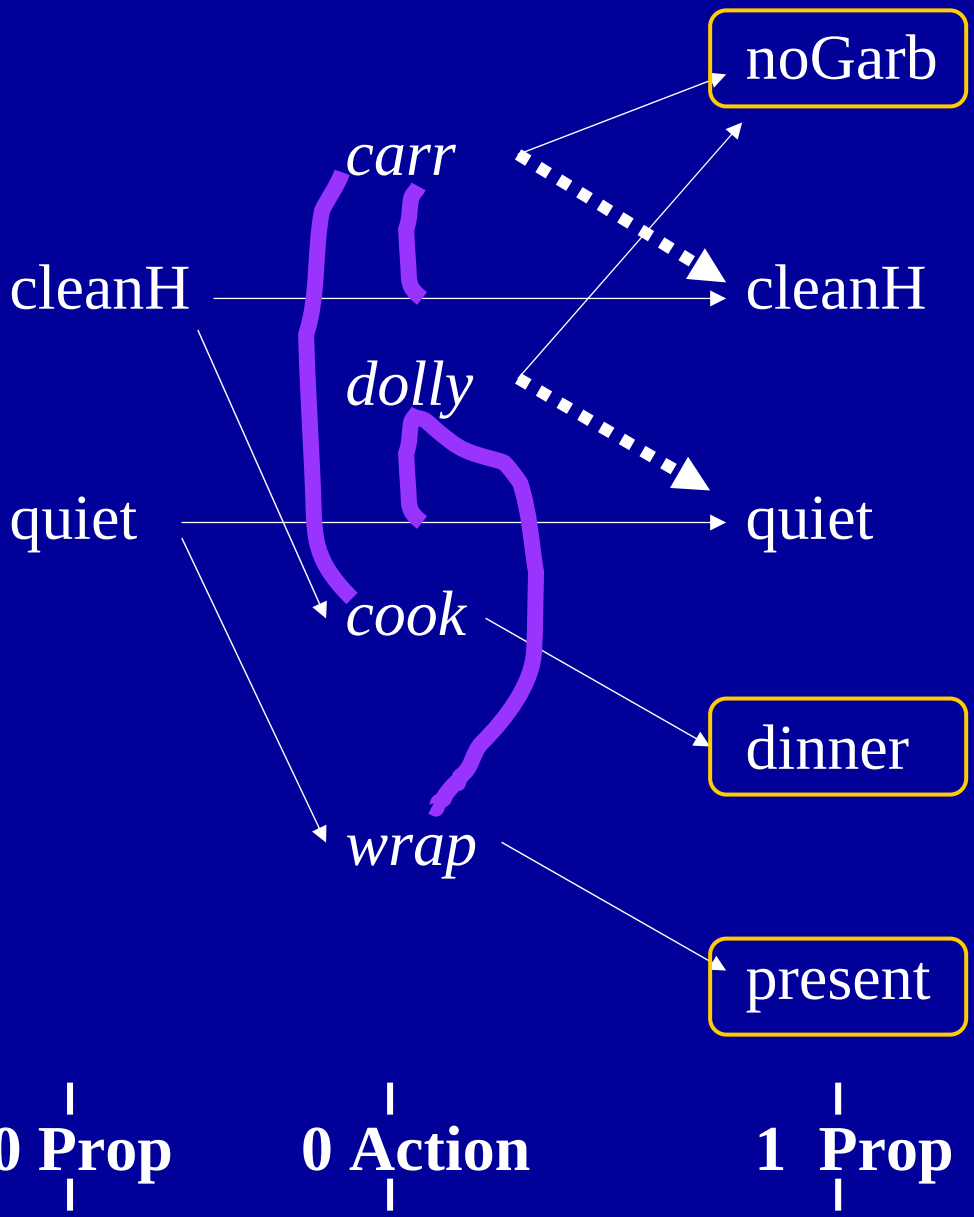


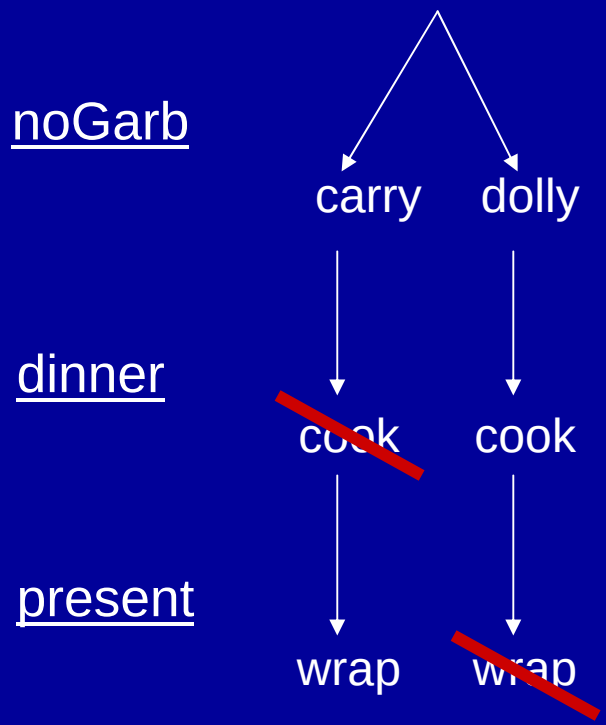
Memos of Inconsistent Subgoals

To prevent wasted search effort:

- If a goal set at layer k cannot be achieved, Then memoize set at layer k.
- Check each new goal set at k against memos.
  - If memo, then fail,
  - Else test by solving a CSP.

Search Layer 0: Record Memo

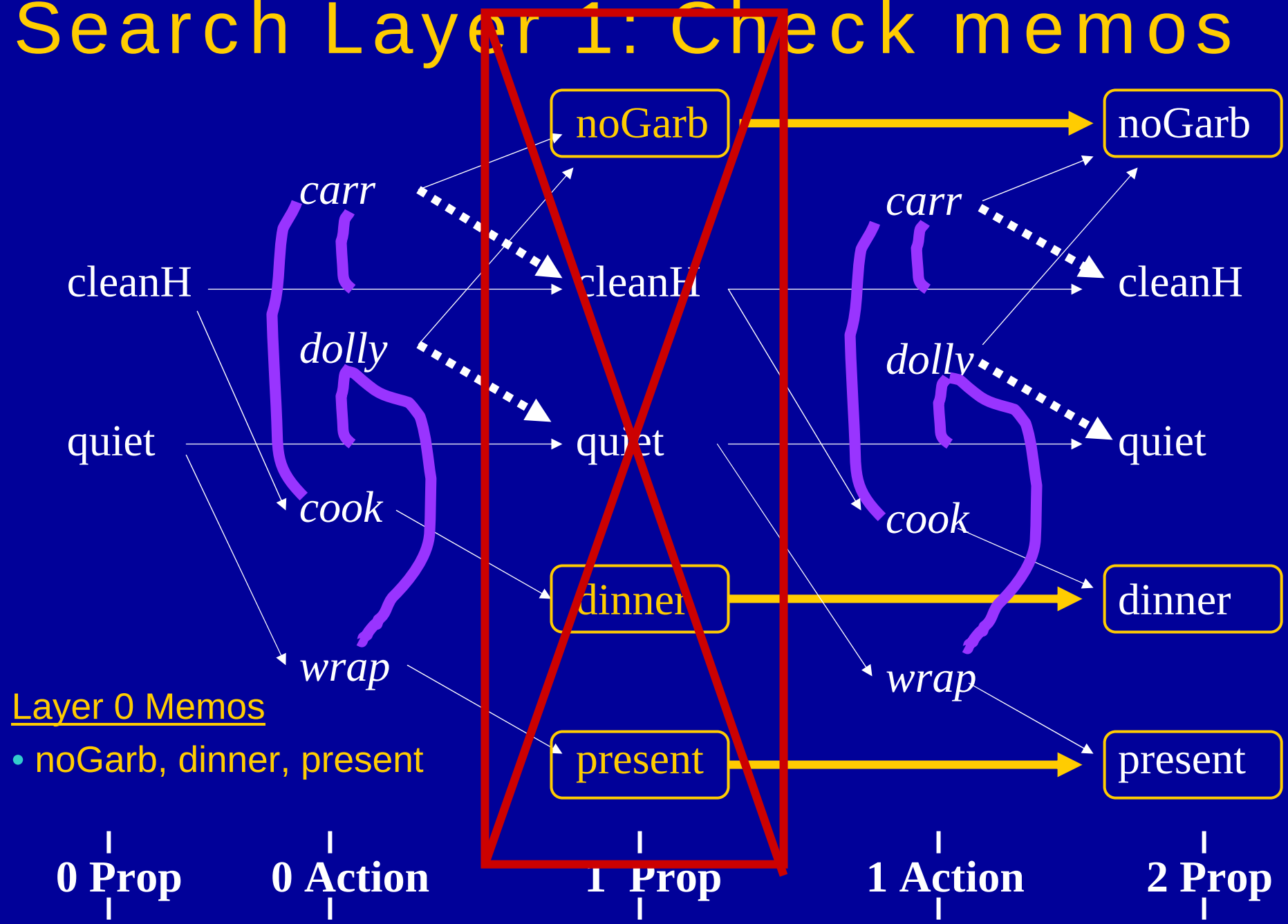




Layer 0 Memos

- noGarb, dinner, present

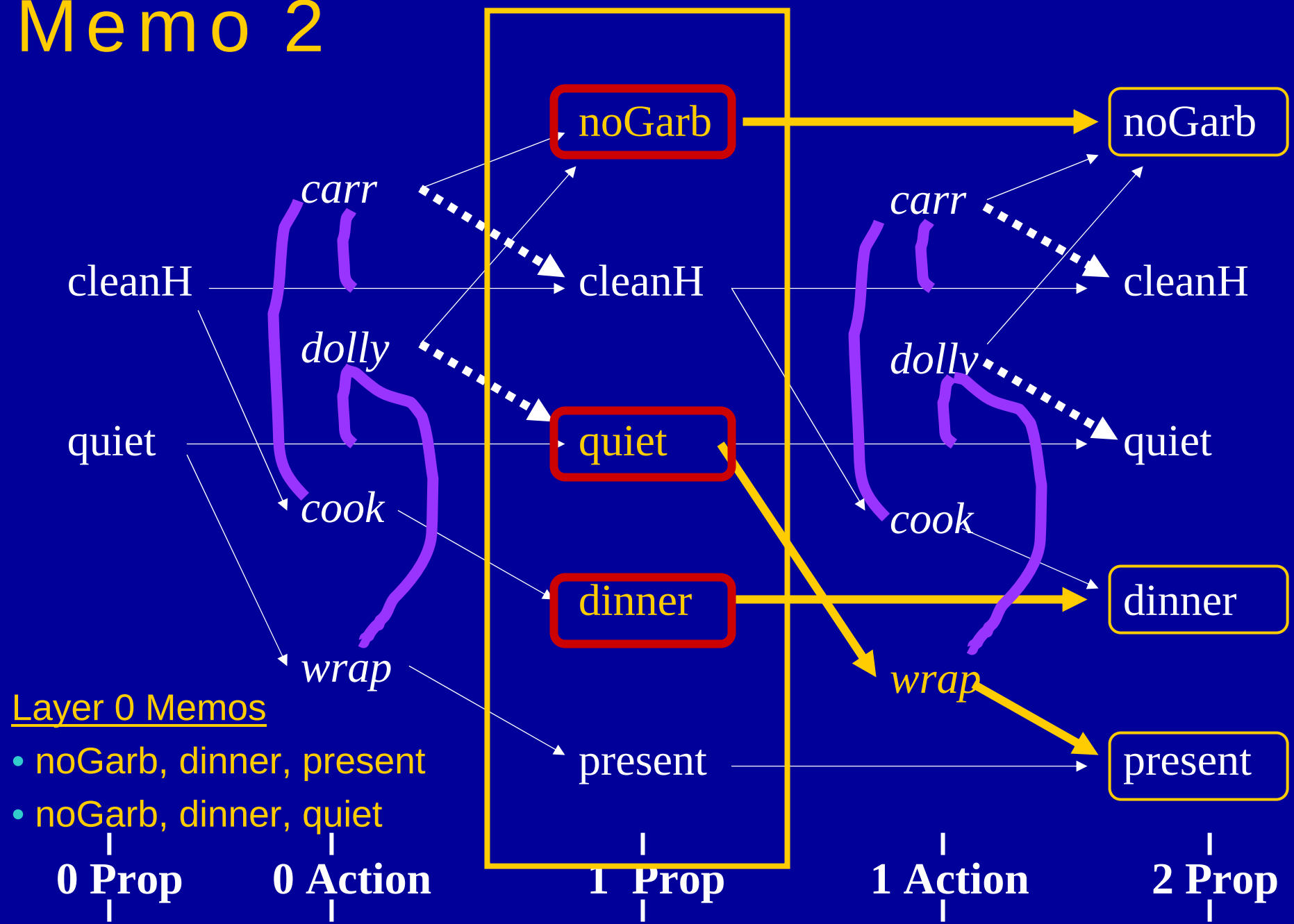
Search Layer 1: Check memos



Layer 0 Memos

- noGarb, dinner, present

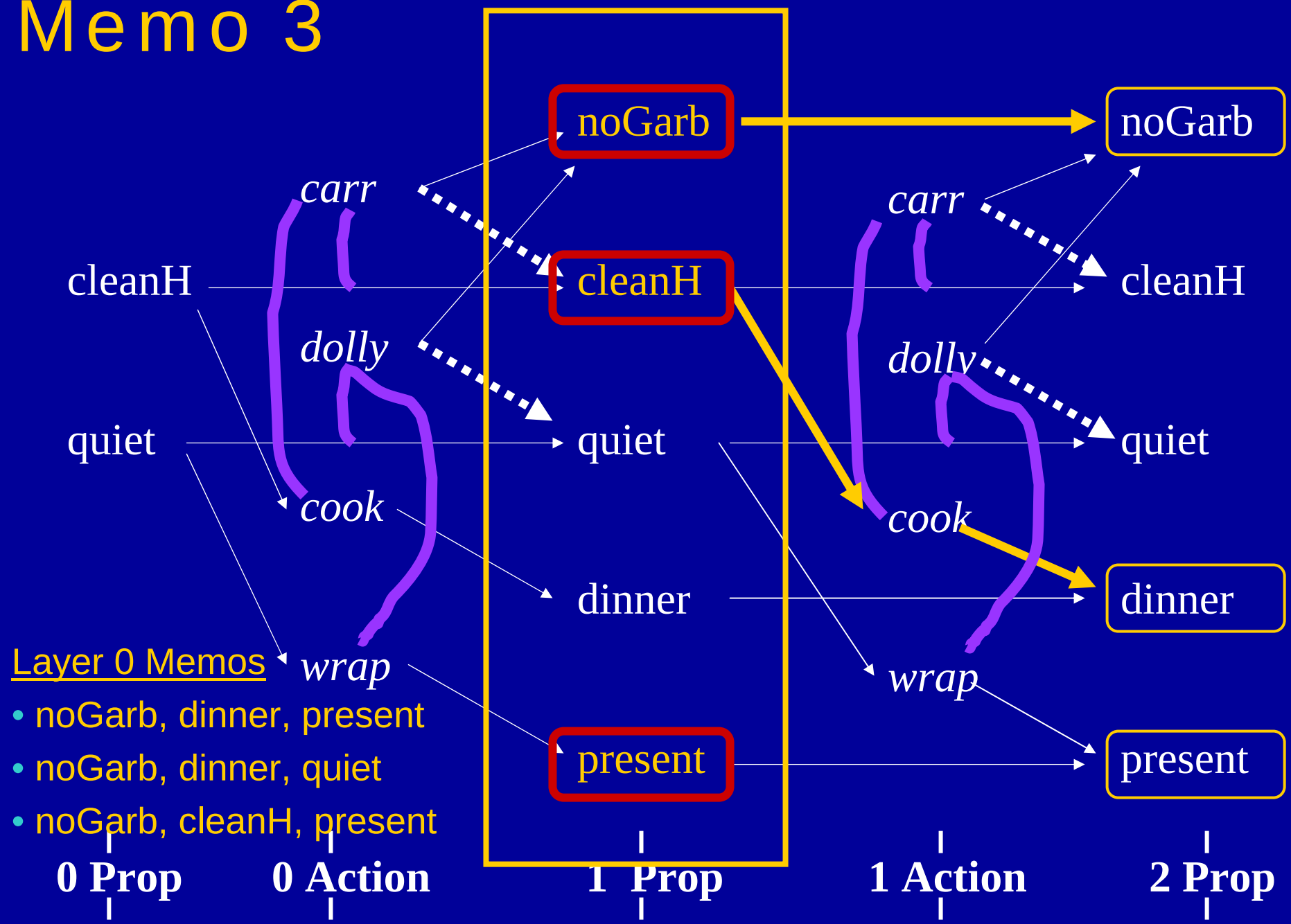
Memo 2



Layer 0 Memos

- noGarb, dinner, present
- noGarb, dinner, quiet

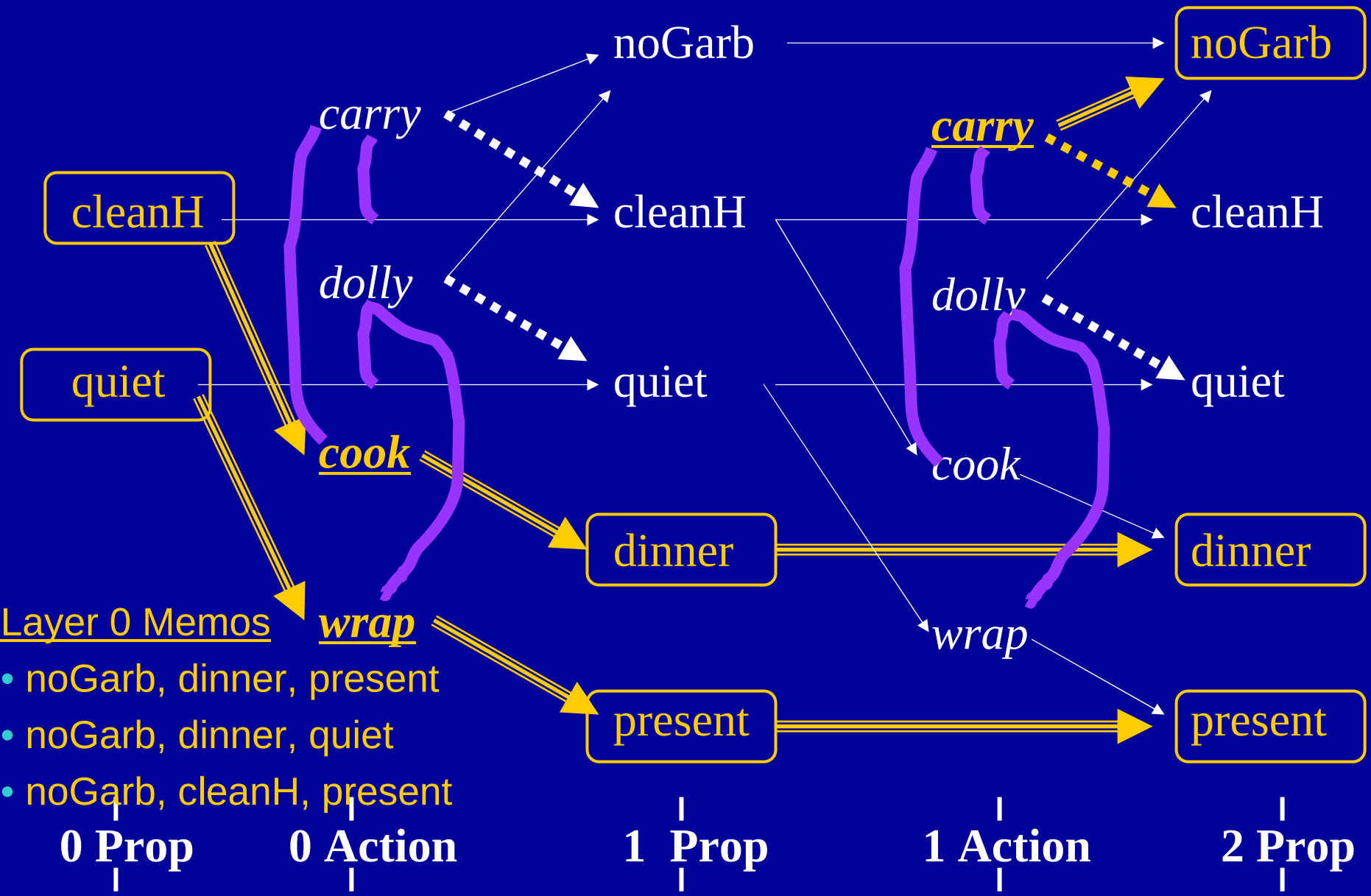
Memo 3



Layer 0 Memos

- noGarb, dinner, present
- noGarb, dinner, quiet
- noGarb, cleanH, present

Solution: Not Recorded as a Memo



Layer 0 Memos

- noGarb, dinner, present
- noGarb, dinner, quiet
- noGarb, cleanH, present

Outline

- Operator-based Planning
- Graph Plan
  - The Graph Plan Planning Problem
  - Graph Construction
  - Solution Extraction
  - Properties
  - Termination with Failure

Plan Graph Properties: Fixed Points

- Propositions monotonically increase
  - once they are added to a layer they are never removed in successive layers;
- Mutexes monotonically decrease
  - once a mutex has decayed it can never reappear;

➔The graph will eventually reach a fix point

- level where facts and mutexs no longer change.

Fix point Example: Door Domain

Move from room 1 to room 2

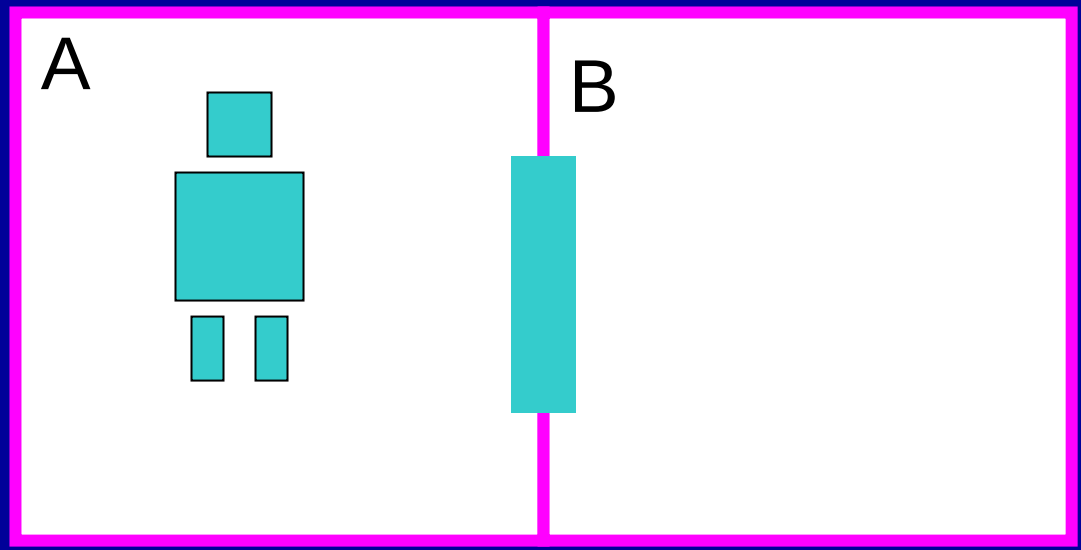
- pre: robot in 1, door is open
- add: robot in 2
- del: robot in 1

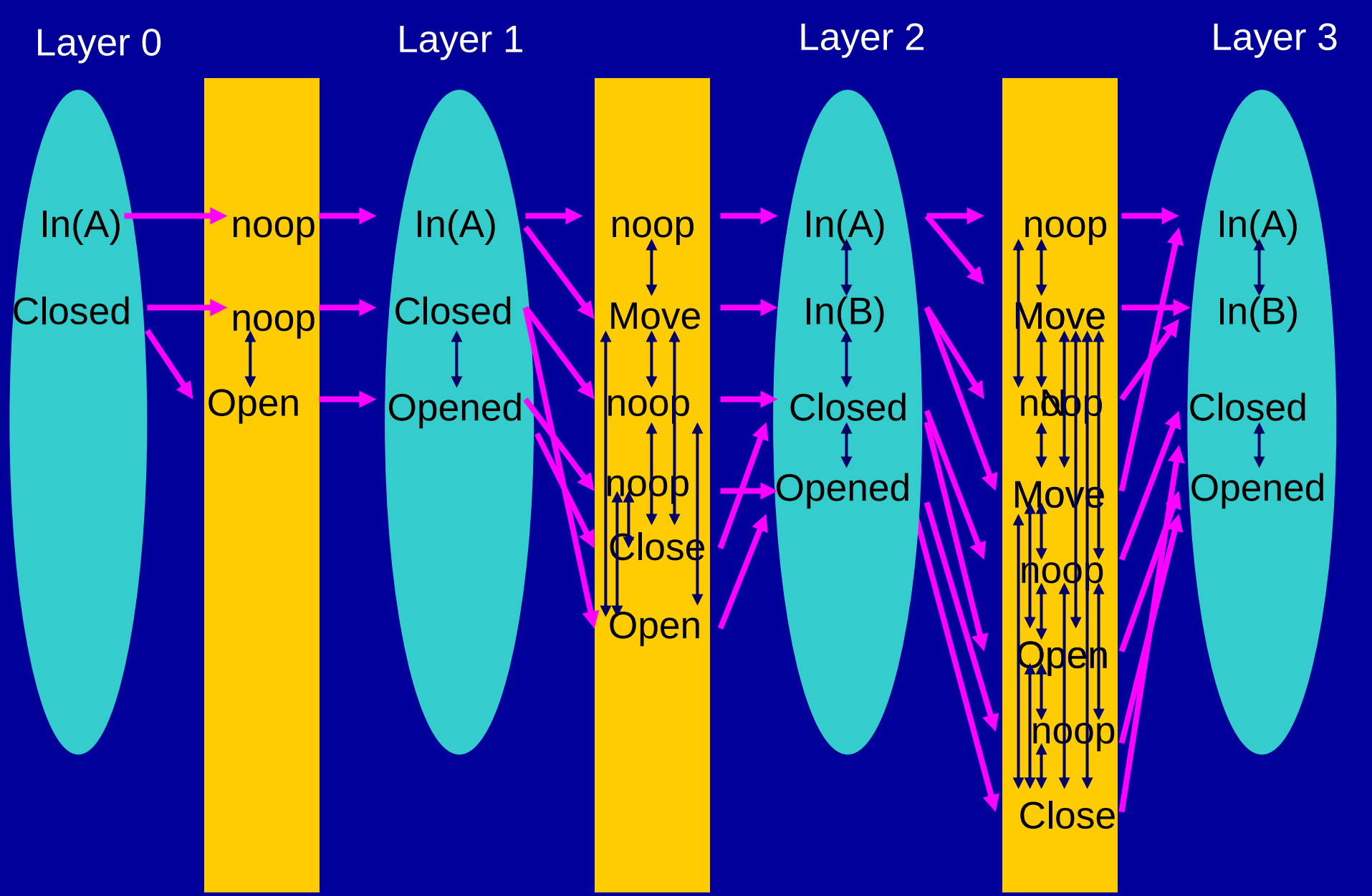
Open door

- pre: door closed
- add: door open
- del: door closed

Close door

- pre: door open
- add: door closed
- del: door open





Layer 3 is the fixed point (called level out) of the graph

Graph Search Properties

- Graphplan may need to expand well beyond the fix point to find a solution.
- Why?

Gripper Example

Move from one room to another

- pre: robot in first room
- add: robot in second room
- del: robot in first room

Pick up ball

- pre: gripper free, ball in room
- add: holding ball
- del: gripper free, ball in room

Drop ball

- pre: holding ball, in room
- add: ball in room, gripper free
- del: holding ball

Gripper Example

- Fix point occurs at layer 4,
  - All facts concerning ball and robot locations are pairwise non-mutex after 4 steps.
- Solution layer depends on # balls moved.
  - E.g., for 30 balls
    - solution is at layer 59,
    - 54 layers with identical facts, actions and mutexes.

Properties: Optimality and Redundancy

- Plans guarantee parallel optimality.
  - Parallel plan will take as short a time as possible.
- Plans don't guarantee sequential optimality.
  - Might be possible to achieve all goals at a later layer using fewer actions.
- Plans do not contain redundant steps.
  - By preferring no-ops.

Outline

- Operator-based Planning
- Graph Plan
  - The Graph Plan Planning Problem
  - Graph Construction
  - Solution Extraction
  - Properties
  - Termination with Failure

Termination Property

- Graphplan returns failure if and only if no plan exists.

Simple Termination

- If the fix point is reached and:
  - A goals is not asserted OR
  - Two goals are mutex

  Then return "No solution," without any search .
- Else may be higher order exclusions (memos), preventing a solution.

➔Requires more sophisticated termination test.

Why Continue After FixPoint?

- propositions, actions and mutexes no longer change after fix point.
- N-ary exclusions (memos) DO change.
  - New layers add time to graph.
  - Time allows actions to be spaced so that memos decay.
  - Memos monotonically decrease
    - Any goal set achievable at layer i, is achievable at i + n.

➔ Track memos & terminate on their fix point.

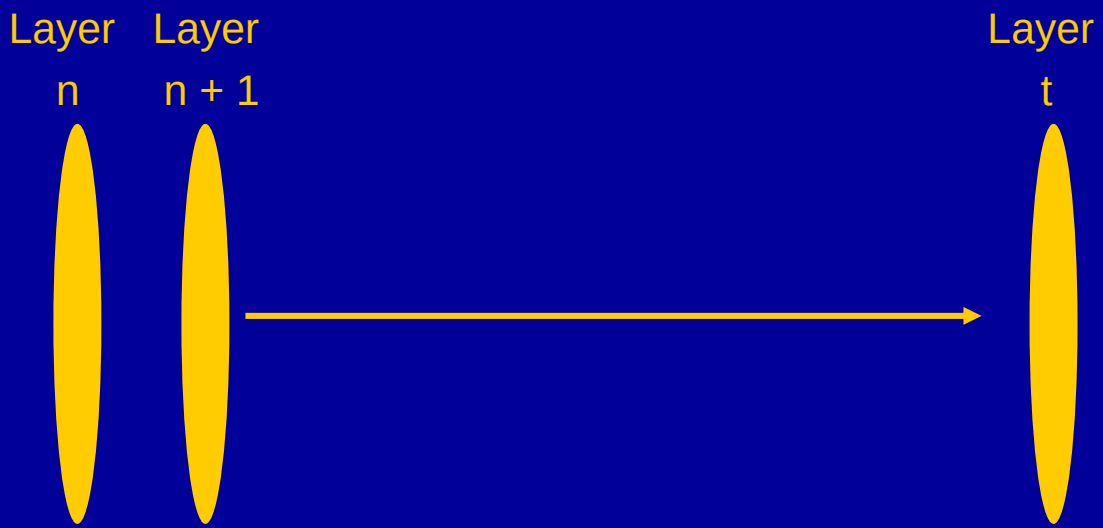
Recap: Graph Plan

- Graphplan developed in 1995 by Avrim Blum and Merrick Furst, at CMU.
- Graphplan searches a compact encoding of the state space, constructed from the operators and initial state.
- Encoding pre-prunes many invalid plans that violate reachability and mutual exclusion.
- Graphplan has been extended to reason with temporally extended actions, metric and non-atomic preconditions and effects.

Appendix

Termination Test

- A graph “levels off” if the memos at layer n+1 are the same as at n.
- If the Graph levels off at layer n, and the current search stage is t >n, Then Graphplan can output "No Solution".



Where S^m_k = the sets of goals found unsolvable at layer k during search from m

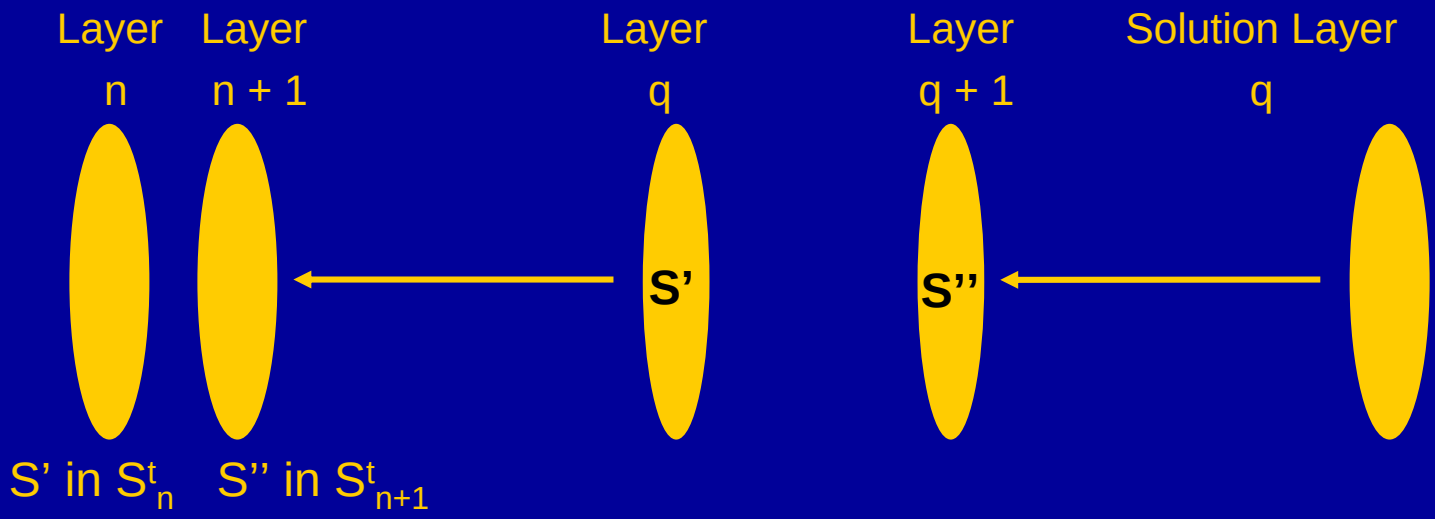
Termination Property

- Theorem: Graphplan returns with failure iff the problem is unsolvable.

- Proof of, If the problem is unsolvable, then Graphplan returns with failure: The number of goal sets found unsolvable at layer n from layer t will never be smaller than the number at n from layer t+1. In addition, there is a finite maximum number of goal sets. Hence, if the problem is unsolvable, eventually two successive layers will contain the same memoized sets.

If Graphplan outputs "No Solution," then the problem is unsolvable.

- Suppose the fix point is at layer n and Graphplan has completed an unsuccessful search starting at layer t > n.
- A plan to achieve any goal set that is unsolvable at layer n+1 must, one step earlier, achieve some set unsolvable at layer n.
- Suppose Graphplan returns "No Solution," but the problem is solvable:



- If $S^t_n = S^t_{n+1}$ then S' and S'' must both be in S^t_{n+1}. This means that some set in S^t_{n+1} will need to be achieved in n+1. this situation is contradictory.