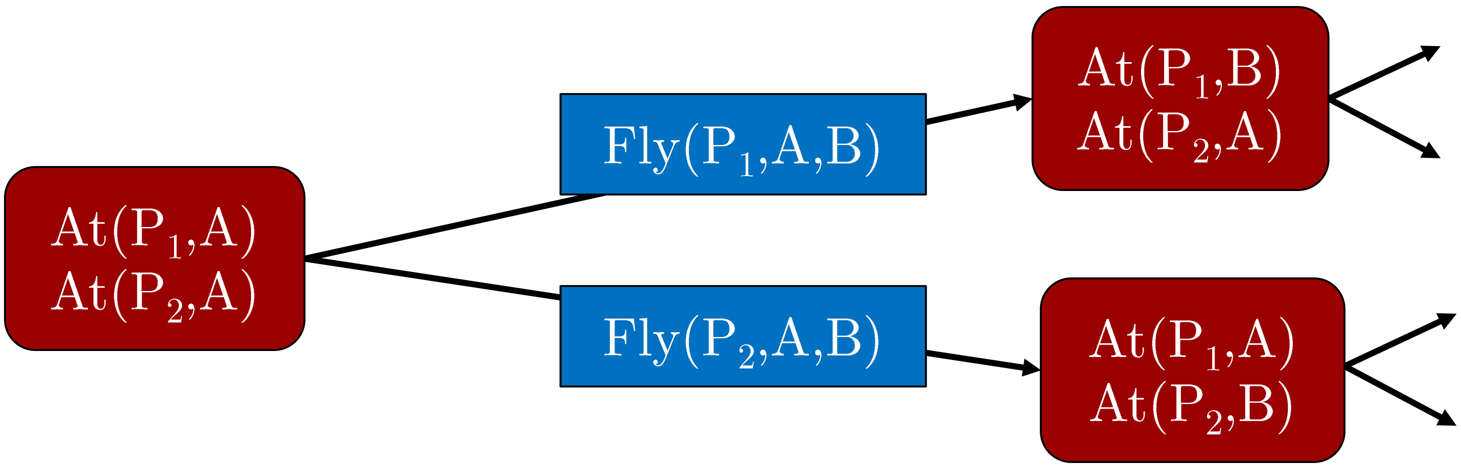
CMU 15-381

Lecture 6:
Planning II

Teachers:
Emma Brunskill
Ariel Procaccia (this time)

Planning as search

- Search from initial state to goal
- Can use standard search techniques, including heuristic search

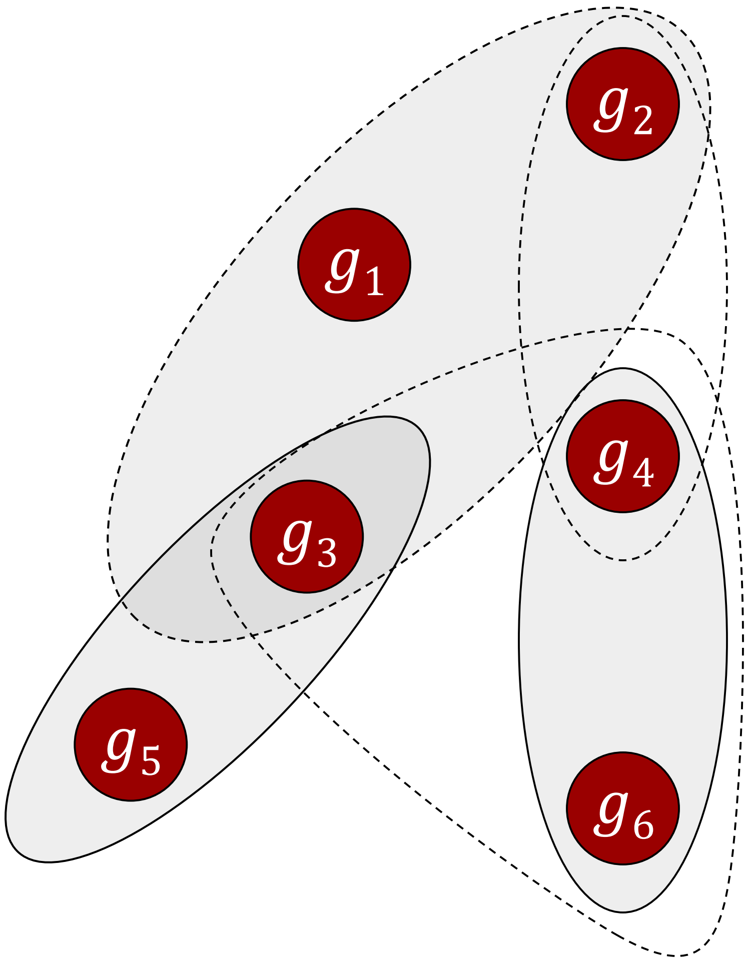


Ignore preconditions

- Heuristic drops all preconditions from operations
- First attempt: # unsatisfied goals
- Complications:
  a. Some operations achieve multiple goals
  b. Some operations undo the effects of others
- Poll 1: To get an admissible heuristic, ignore:
  1. Just a
  2. Just b
  3. Both a and b

Ignore preconditions

- Count min number of operations s.t. the union of their effects contains goals
- This is the Set Cover problem: NP-hard!
- Approximation is:
    - Also hard!
    - Inadmissible!



Ignore preconditions

- Possible to ignore specific preconditions to derive domain-specific heuristics
- Sliding block puzzle;
- $\text{On}(t, s_1) \wedge \text{Blank}(s_2) \wedge \text{Adjacent}(s_1, s_2) \Rightarrow \text{On}(t, s_2) \wedge \text{Blank}(s_1) \wedge \neg\text{On}(t, s_1) \wedge \neg\text{Blank}(s_2)$
- Consider two options:
  a. Removing $\text{Blank}(s_2) \wedge \text{Adjacent}(s_1, s_2)$
  b. Removing $\text{Blank}(s_2)$
- Poll 2: Match option to heuristic:
  1. a↔ ∑Manhattan, b↔#misplaced tiles
  2. a↔#misplaced tiles, b↔ ∑Manhattan
  3. b↔#misplaced tiles, a is inadmissible
  4. b↔ ∑Manhattan, a is inadmissible

5	2	
6	1	3
7	8	4

Example state

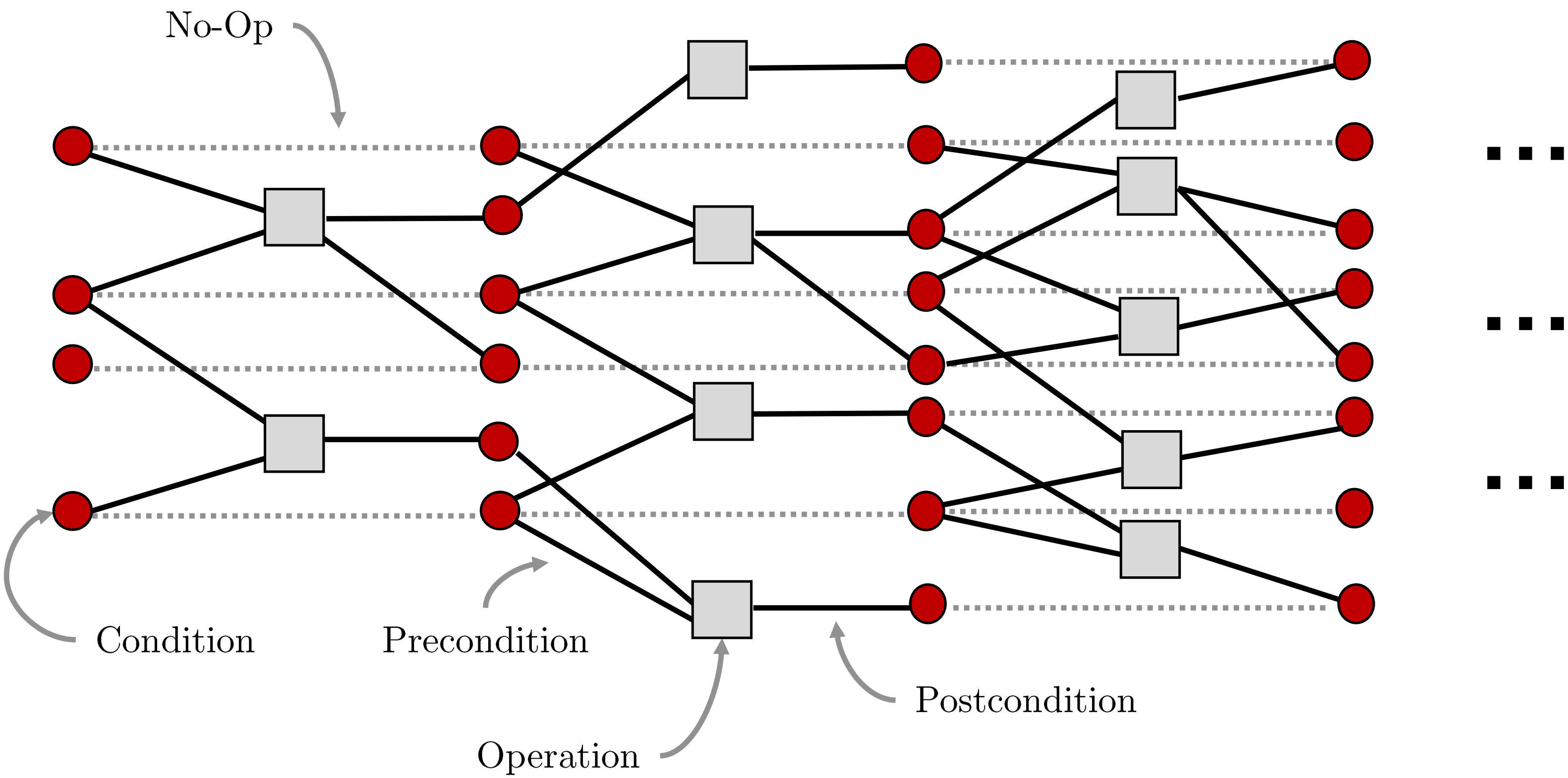
1	2	3
4	5	6
7	8	

Goal state

Planning graphs

- Leveled graph: vertices organized into levels, with edges only between levels
- Two types of vertices on alternating levels:
    - Conditions
    - Operations
- Two types of edges:
    - Precondition: condition to operation
    - Postcondition: operation to condition

Generic planning graph*



* Slide based on Brafman

Graph construction

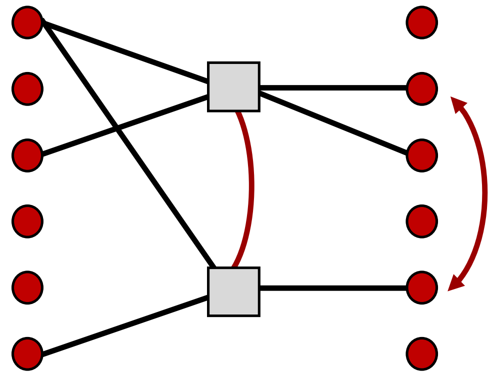
- S_0 contains conditions that hold in initial state
- Add operation to level O_i if its preconditions appear in level S_i
- Add condition to level S_i if it is the effect of an operation in level O_{i-1} (no-op action also possible)
- Idea: S_i contains all conditions that could hold at time i; O_i contains all operations that could have their preconditions satisfied at time i
- Can optimistically estimate how many steps it takes to reach a goal

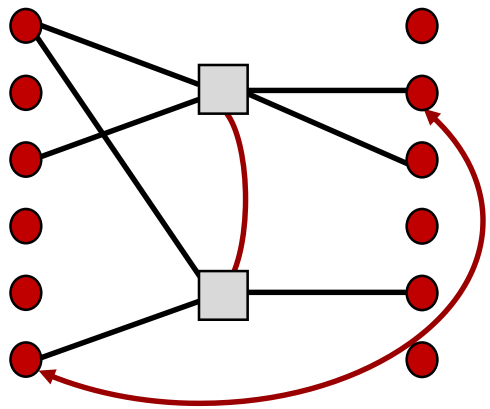
Mutual exclusion

- Two operations or conditions are mutually exclusive (mutex) if no valid plan can contain both
- A bit more formally:
  - Two operations are mutex if their preconditions or postconditions are mutex
  - Two conditions are mutex if one is the negation of the other, or all actions that achieve them are mutex
- Even more formally…

Mutex cases*

- Inconsistent postconditions (two ops): one operation negates the effect of the other
- Interference (two ops): a postcondition of one operation negates precondition of the other

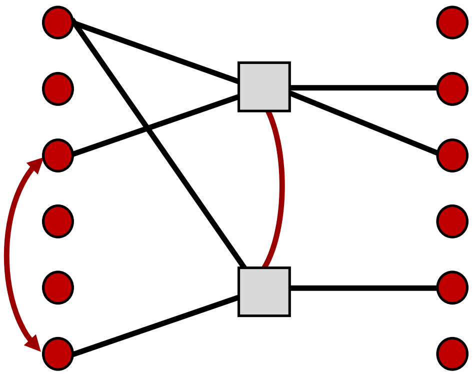

Inconsistent Postconditions


Interference

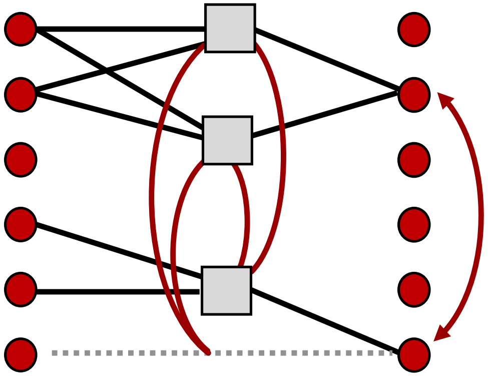
* Slide based on Brafman

Mutex cases*

- Competing needs (two ops): a precondition of one operation is mutex with a precondition of the other
- Inconsistent support (two conditions): every possible pair of operations that achieve both conditions is mutex



Competing Needs

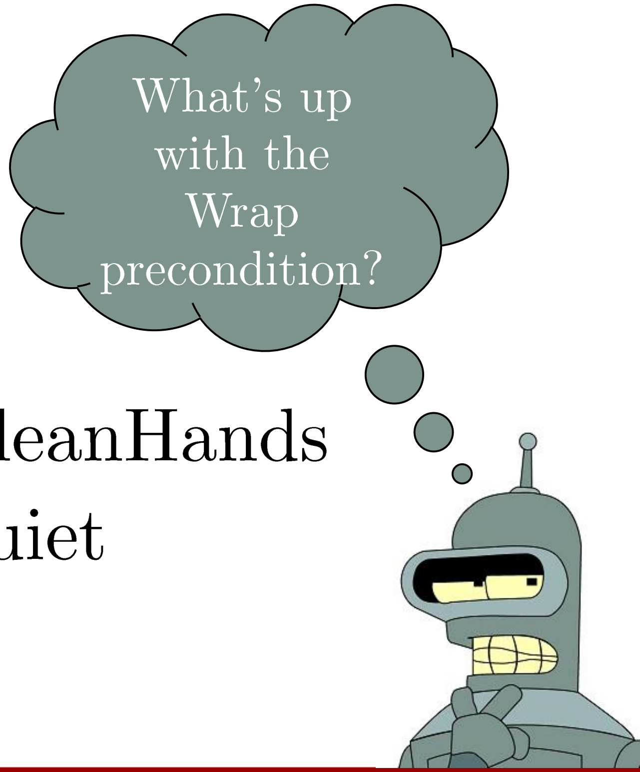


Inconsistent Support

* Slide based on Brafman

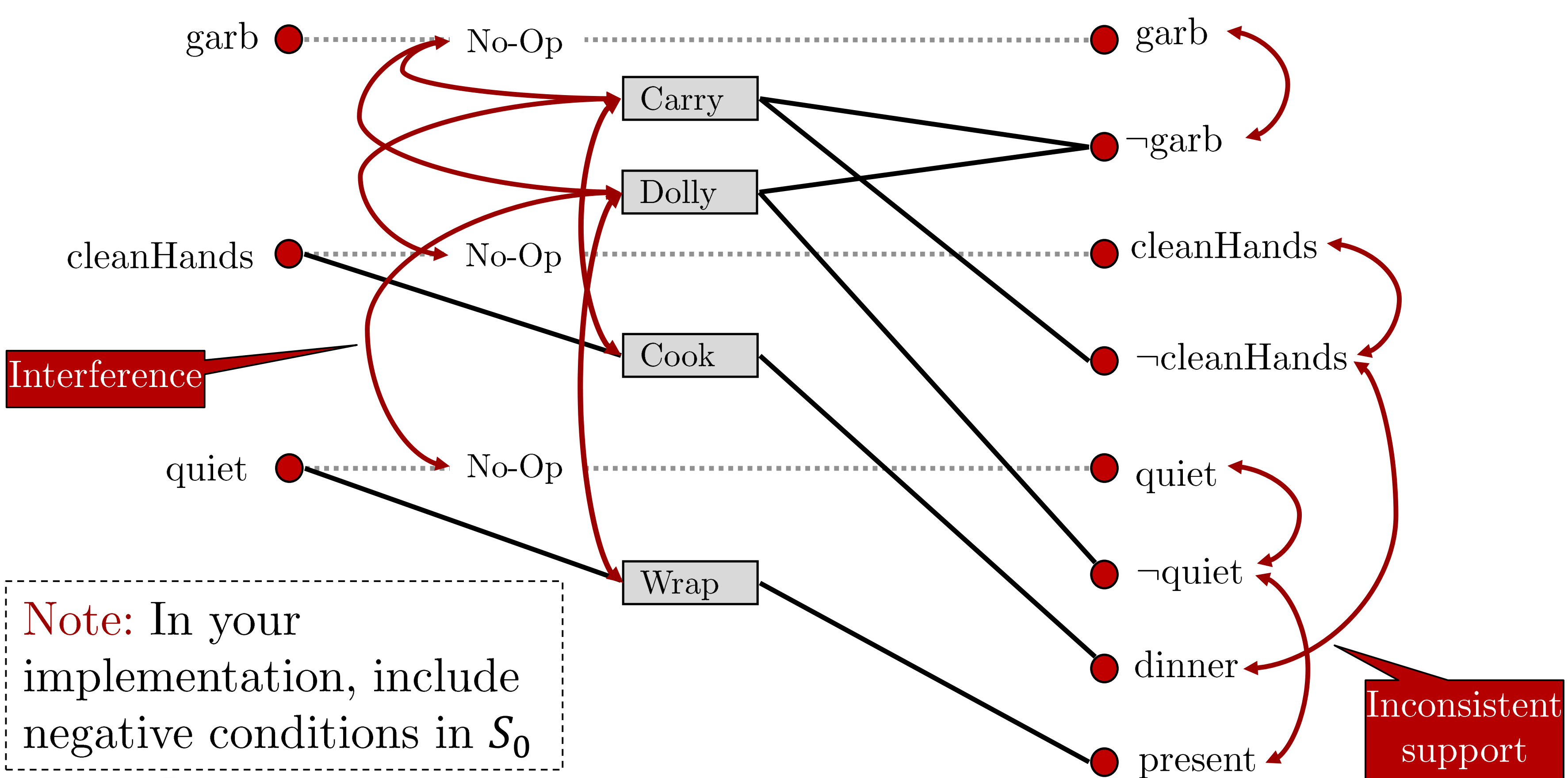
Dinner date example

- Initial state:
  garbage ∧ cleanHands ∧ quiet
- Goals:
  dinner ∧ present ∧ ¬garbage
- Actions:
  - Cook: cleanHands ⇒ dinner
  - Carry: none ⇒ ¬garbage ∧ ¬cleanHands
  - Dolly: none ⇒ ¬garbage ∧ ¬quiet
  - Wrap: quiet ⇒ present



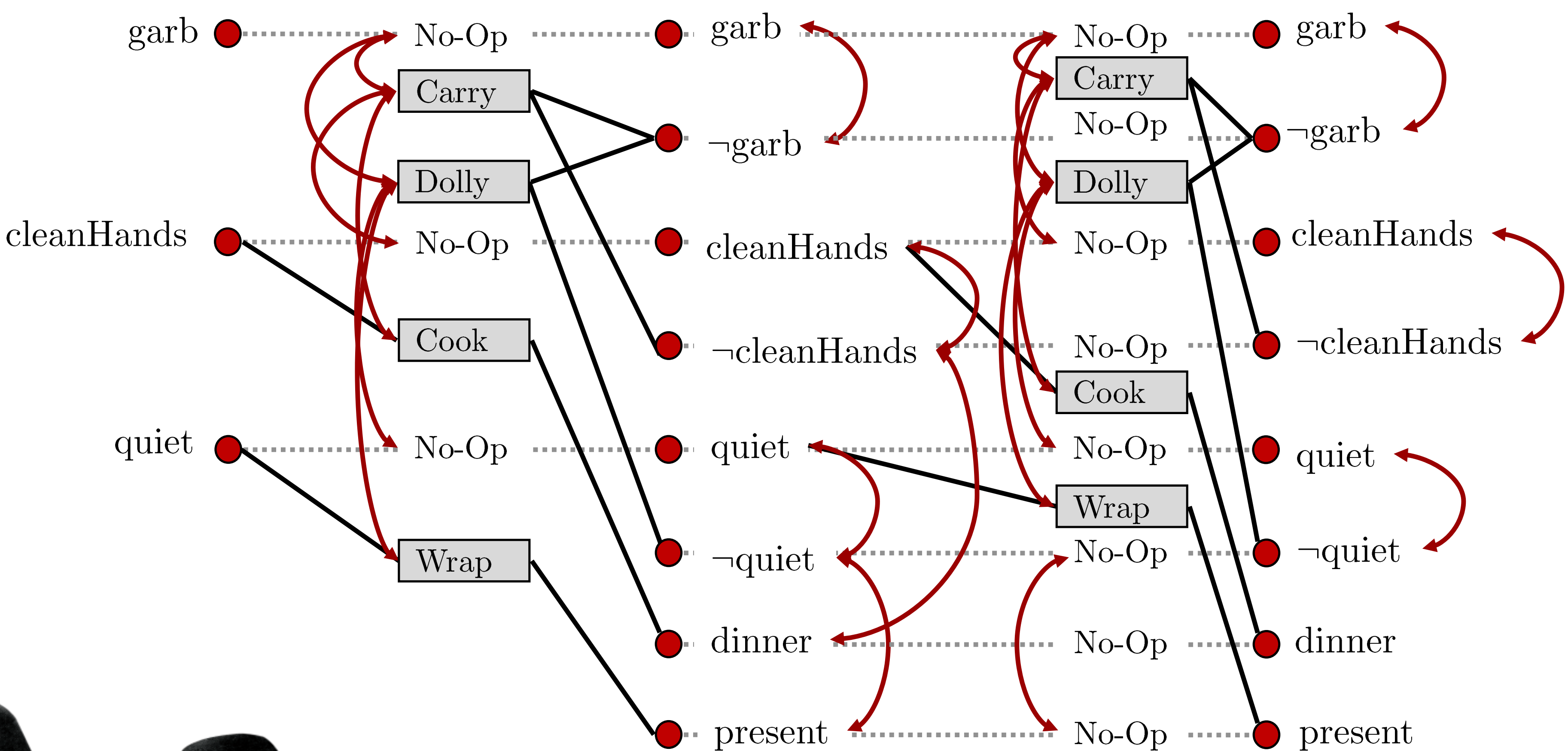
* Slide based on Brafman

Dinner date example*



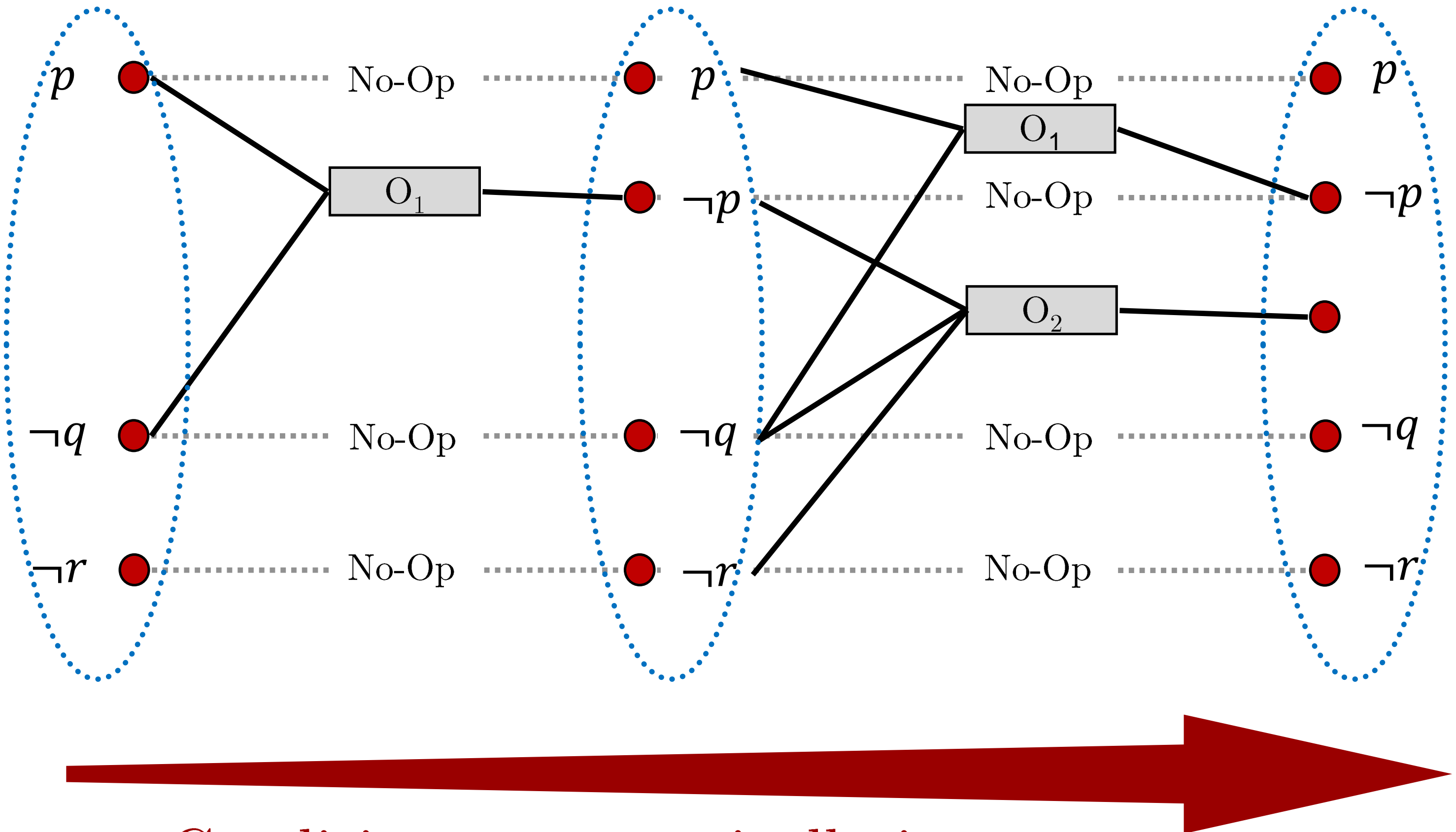
Note: In your implementation, include negative conditions in S_0

Dinner date example*



* Slide based on Brafman

Observation 1*

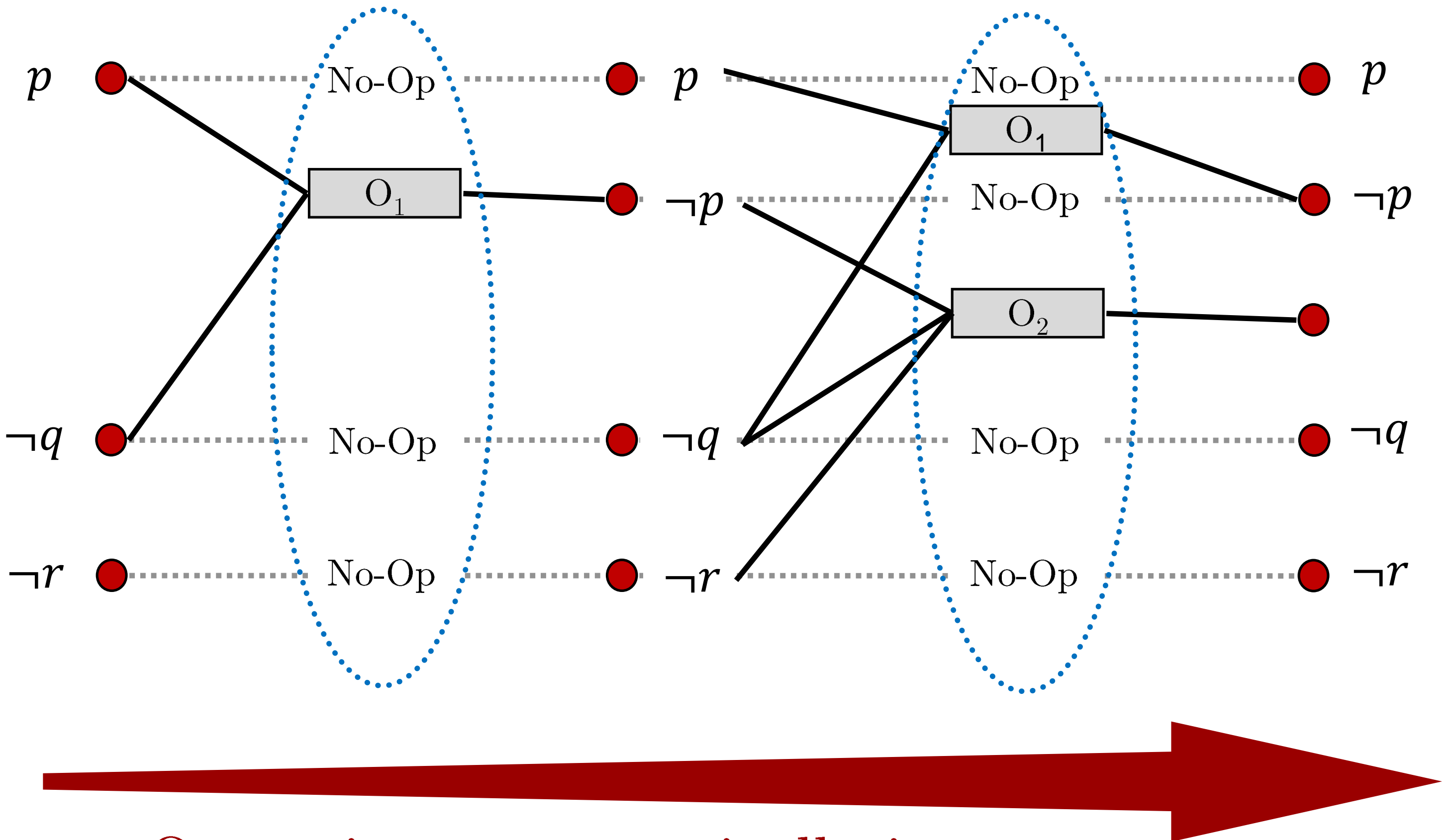


Conditions monotonically increase

(always carried forward by no-ops)

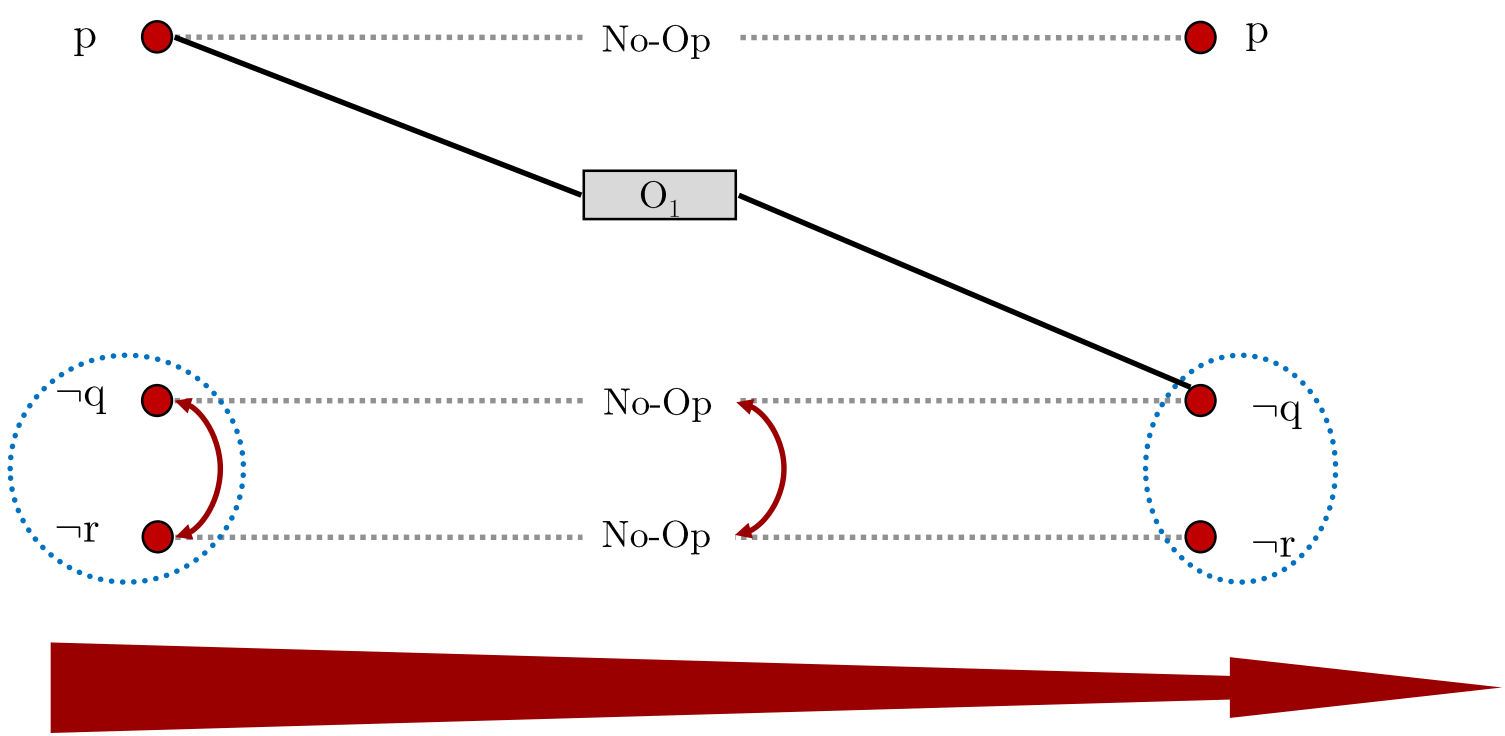
* Slide based on Brafman

Observation 2*



Operations monotonically increase

Observation 3*



Condition mutex relationships monotonically decrease

* Slide based on Brafman

OBSERVATION 4

- Operation mutexes monotonically decrease
- Proof idea:
  - Inconsistent postconditions and interference are properties of the operations themselves $\Rightarrow$ hold at every level
  - Competing needs: condition mutexes are monotonically decreasing
- To be formal, need to do a double induction on proposition and operation mutexes

Leveling off

- As a corollary of the observations, we see that the planning graph levels off
  - Consecutive levels become identical
- Proof:
  - Upper bound on #operations and #conditions
  - Lower bound of 0 on #mutexes ■

Heuristics from graphs

- Level cost of goal g = level where g first appears
- To estimate the cost of all goals:
  - Max level: max level cost of any goal (clearly admissible)
  - Level sum: sum of level costs
  - Set level: level at which all goals appear without any pair being mutex
- Poll 3: Which is admissible:
  1. Level sum
  2. Set level
  3. Both
  4. Neither

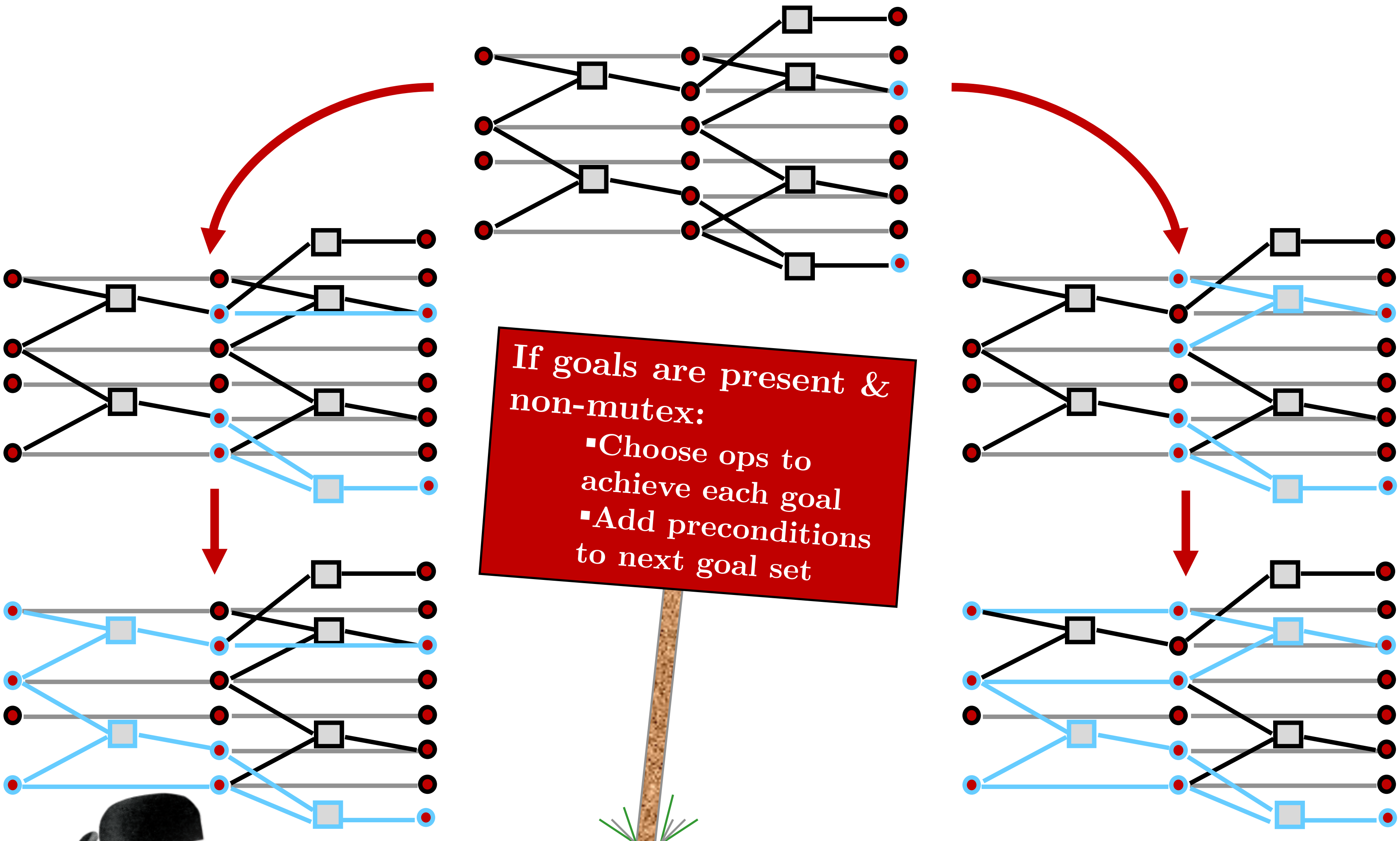
The Graphplan algorithm

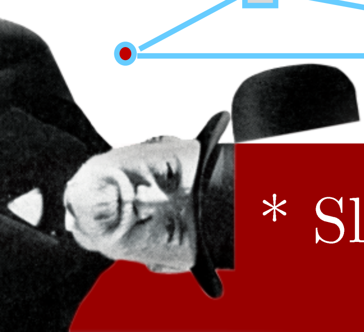
1. Grow the planning graph until all goals are reachable and not mutex
   (If planning graph levels off first, fail)
2. Call Extract-Solution on current planning graph
3. If none found, add a level to the planning graph and try again

Extract-Solution

- Search where each state corresponds to a level and a set of unsatisfied goals
- Initial state is the last level of the planning graph, along with the goals of the planning problem
- Actions available at level S_i are to select any conflict-free subset of operations in A_{i-1} whose effects cover the goals in the state
- Resulting state has level S_{i-1} and its goals are the preconditions for selected actions
- Goal is to reach a state at level S_0

EXTRACT-SOLUTION ILLUSTRATED*





* Slide based on Brafman

Graphplan guarantees

- Observation: The size of the t-level planning graph and the time to create it are polynomial in t, #operations, #conditions
- Theorem: Graphplan returns a plan if one exists, and returns failure if one does not exists

Summary

- Terminology:
  - Planning graphs
  - Set level heuristic
- Algorithms:
  - Graphplan
- Big ideas:
  - Planning is search, but admits domain-independent heuristics