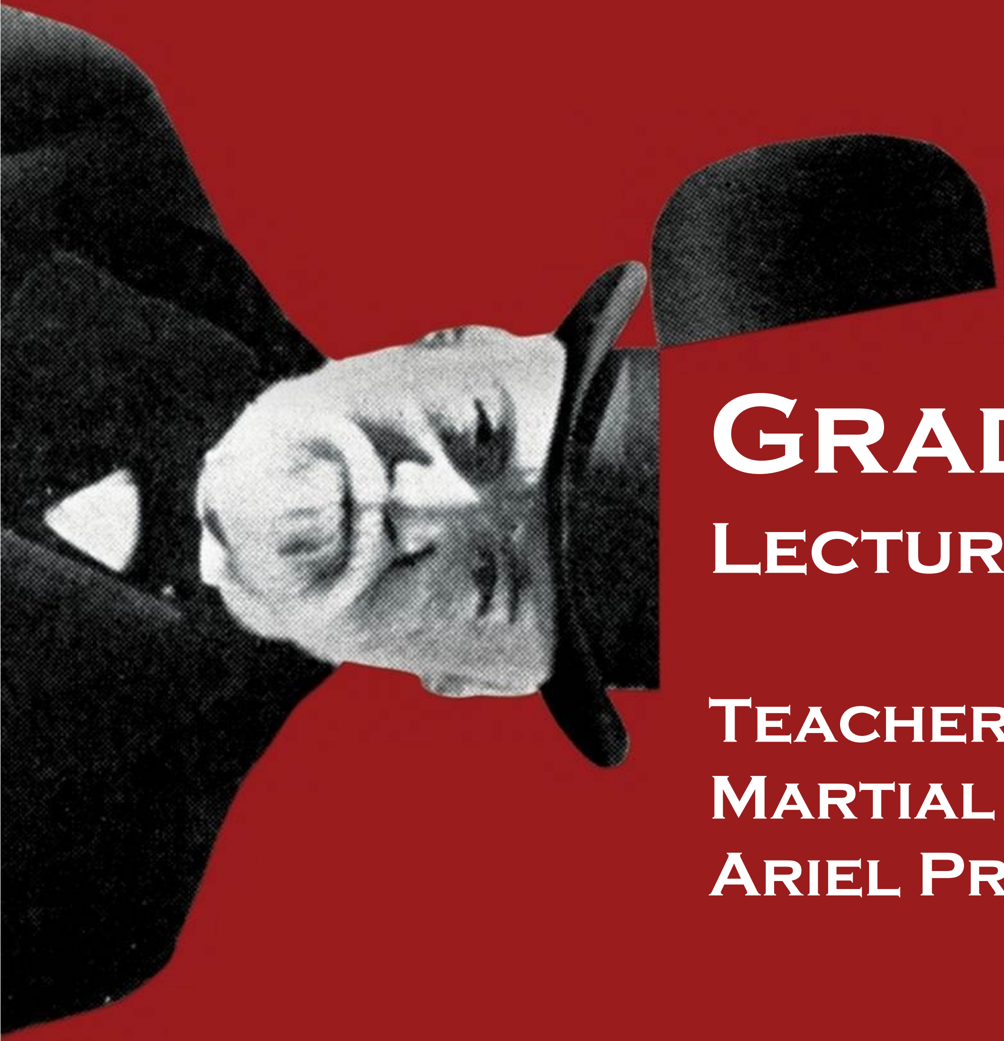


Graduate AI

Lecture 16: Planning 2

Teachers:
Martial Hebert
Ariel Procaccia (this time)

Reminder

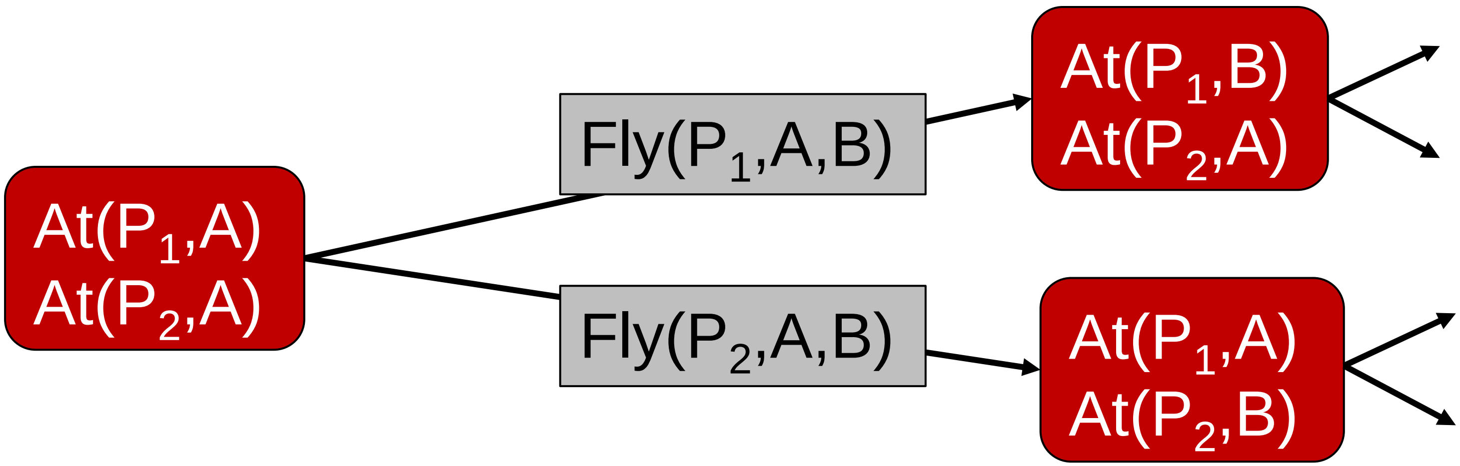
- State is a conjunction of **conditions**, e.g., at(Truck_1,Shadyside)$\wedge$at(Truck_2,Oakland)
- States are transformed via **operators** that have the form
  Preconditions $\Rightarrow$ Effects (postconditions)

Reminder

- Pre is a conjunction of positive and negative conditions that must be satisfied to apply the operation
- Effect is a conjunction of positive and negative conditions that become true when the operation is applied
- We are given the initial state
- We are also given the **goals**, a conjunction of positive and negative conditions

Planning as search

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



Potential obstacles

- Example: inefficient search
  - Operation Buy(isbn) with no preconditions and effect Own(isbn) for each of the 10 billion ISBN numbers
  - Uninformed search must enumerate all options
- Example: large state space
  - 10 airports, each has 5 planes and 20 pieces of cargo
  - Goal: move the cargo at airport A to B
  - Search graph up to the depth of the obvious solution can have $> 10^{100}$ nodes
- From 1961 to 1998 forward search was considered too inefficient to be practical

Backward search

- Searching backward from goal to initial state
- Can help in the examples
- Hard to come up with **heuristics** $\Rightarrow$ modern systems use forward search with killer heuristics

Heuristics for planning

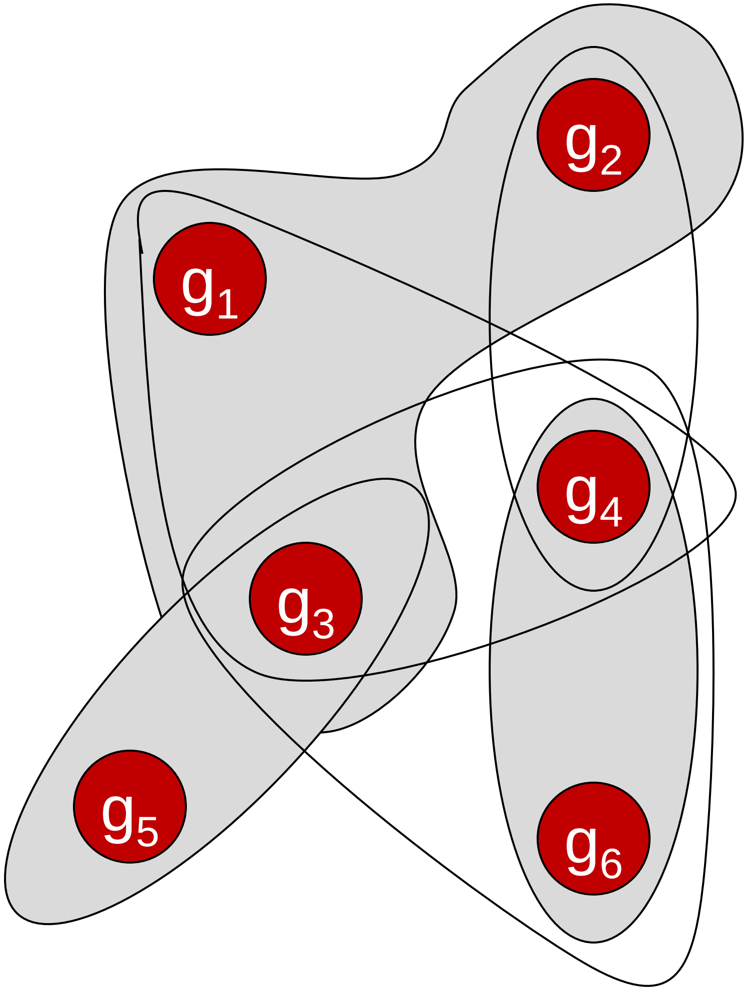
- Define a relaxed problem that is easier to solve and gives an admissible heuristic
- Two general approaches: add edges to the search graph or group multiples nodes together

Ignore preconditions

- Heuristic drops all preconditions from operations
- Any goal condition can be achieves in one step
- Complications:
  1. Some operations achieve multiple goals
  2. Some operations undo the effects of others
- Ignore 2 but not 1: remove preconditions and all effects except goal conditions
- Count min number of operations s.t. the union of their effects contains goals

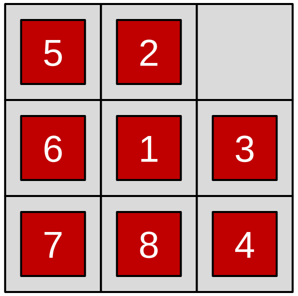
Set cover

- This is exactly the set cover problem
- Problem is NP-hard
- Hard to approximate to a factor better than logn
- Approximation is inadmissible

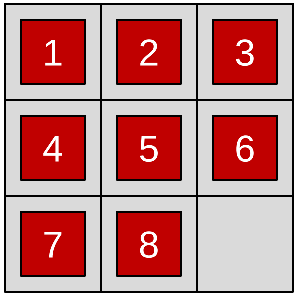


Ignore preconditions

- Possible to ignore *specific* preconditions
- Sliding block puzzle;
- $On(t,s_1) \land Blank(s_2) \land Adjacent(s_1,s_2) \Rightarrow On(t,s_2) \land Blank(s_1) \land \neg On(t,s_1) \land \neg Blank(s_2)$
- Removing $Blank(s_2) \land Adjacent(s_1,s_2)$ gives…
  - #misplaced tiles heuristic
- Removing $Blank(s_2)$ gives…
  - Manhattan distance heuristic
- Can derive domain-specific heuristics



Example state

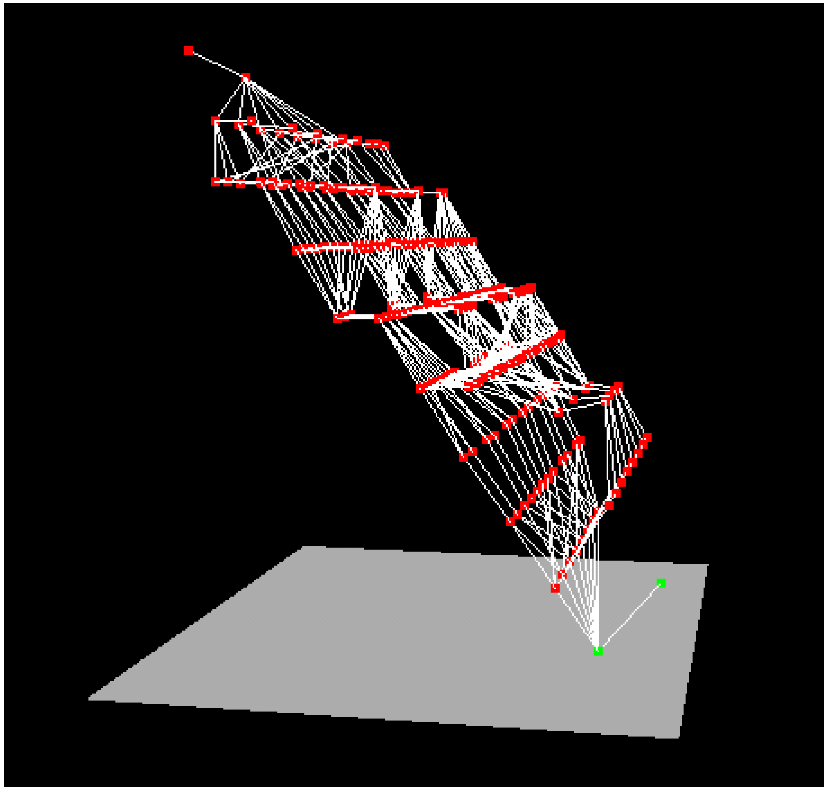


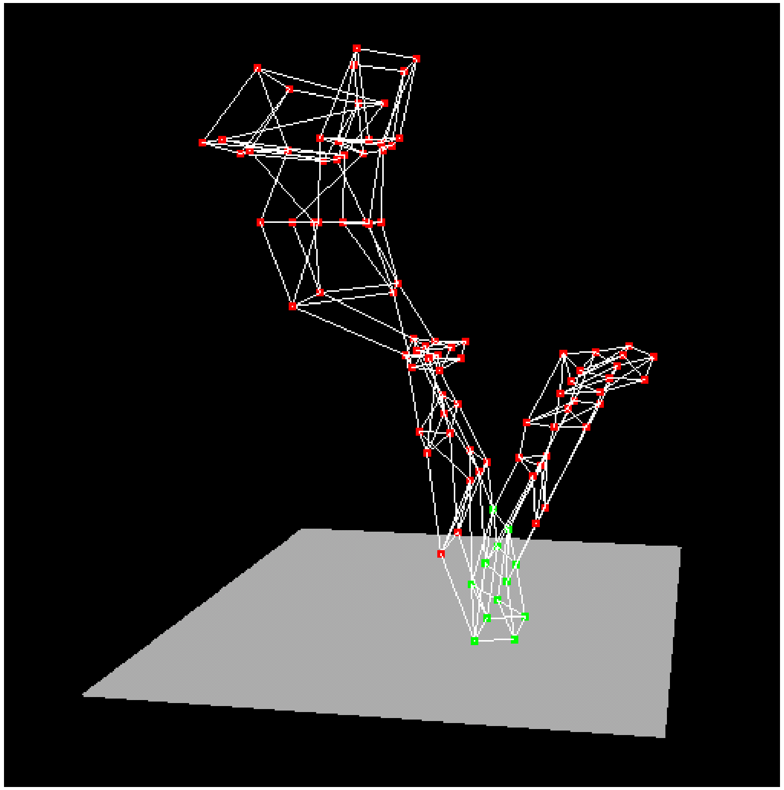
Goal state

Ignore delete lists

- Assume that goals and preconditions contain only positive literals
  - Can rewrite if not
- Remove delete lists from all operations
- Make monotonic progress towards goals
- Still NP-hard to find a solution (proved in lecture 15, slide 19)
  - Why doesn't this follow from NP-hardness of set cover?
- "Hill-climbing" works well

Hill climbing





Hoffman, JAIR 2005

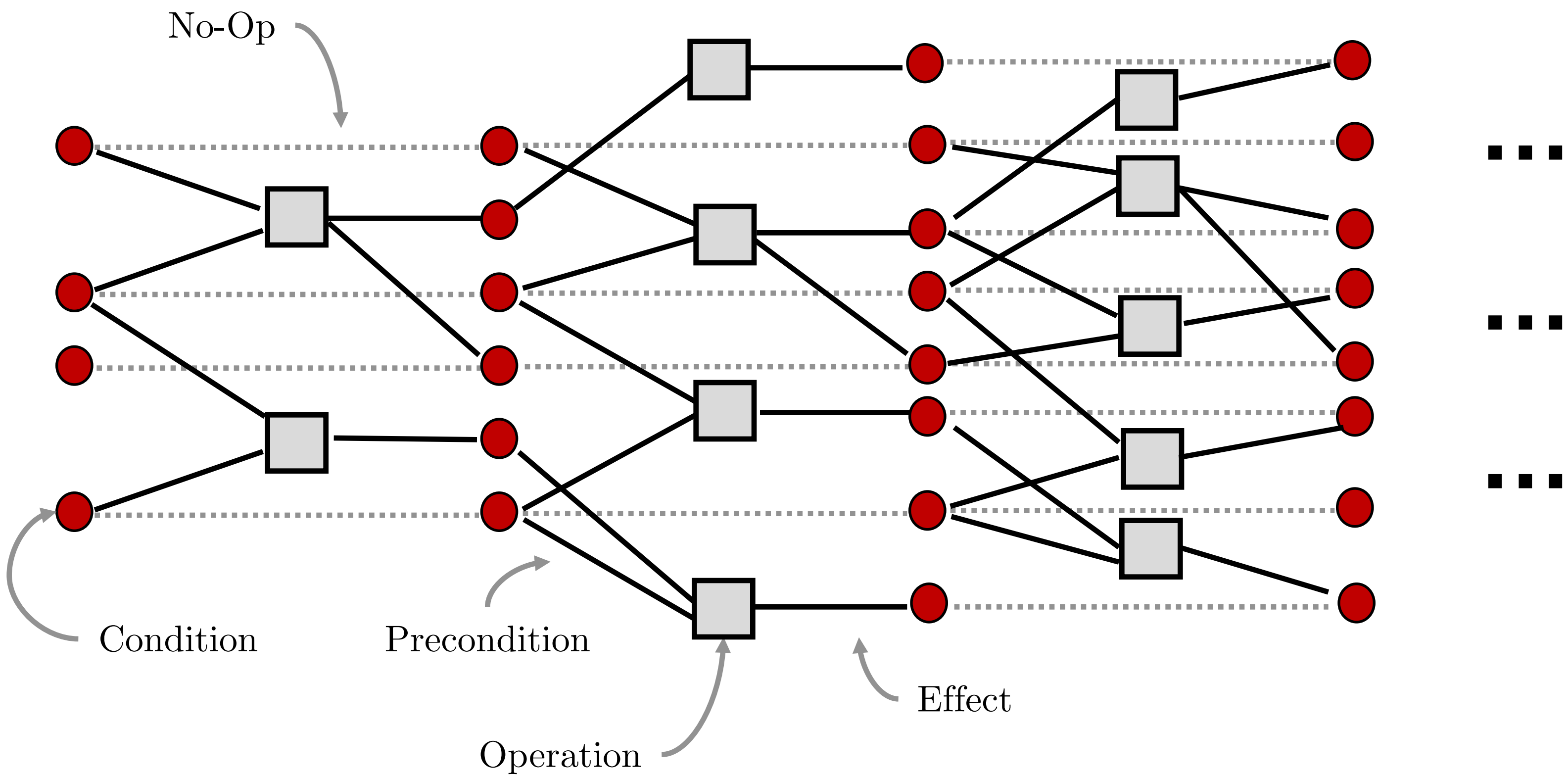
State abstraction

- Relaxed problem is still an expensive way to compute a heuristic if there are many states
- Consider air cargo problem with 10 airports, 50 planes, 200 pieces of cargo
- $\#\text{states} = 10^{50} \times (50+10)^{200} > 10^{250}$
- Assume all packages are in 5 airports, packages in airport have the same destination
  $\Rightarrow$ 5 planes and 5 packages
- $\#\text{states} = 10^{5} \times (5+10)^{5} < 10^{11}$

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
  - Effect: operation to condition

Generic planning graph



Graph construction

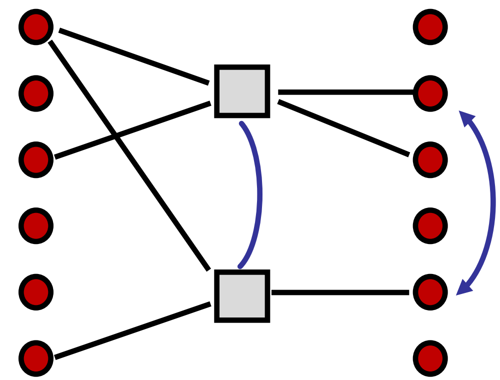
- S_0 contains conditions that hold in initial state
- Add operation to level O_i if its preconditions appear in level S_i
- Add condition to level S_i if it is the effect of an operation in level O_{i-1} (no-op action also possible)
- Idea: S_i contains all conditions that could hold at time i; O_i 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

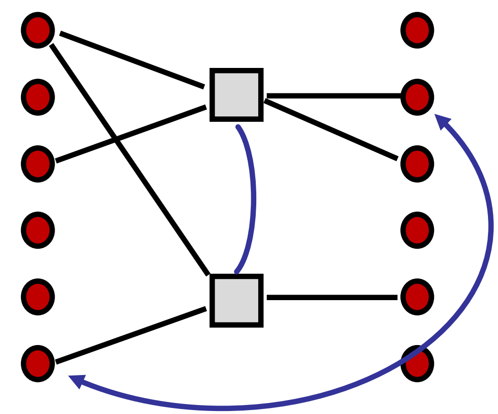
- Two operations or conditions are mutually exclusive (mutex) if no valid plan can contain both
- A bit more formally:
  - Two operations are mutex if their preconditions or effects are mutex
  - Two conditions are mutex if one is the negation of the other, or all actions that achieve them are mutex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

Mutex cases

- Inconsistent effects (two ops): one operation negates the effect of the other
- Interference (two ops): an effect of one operation negates precondition of other



Inconsistent Effects

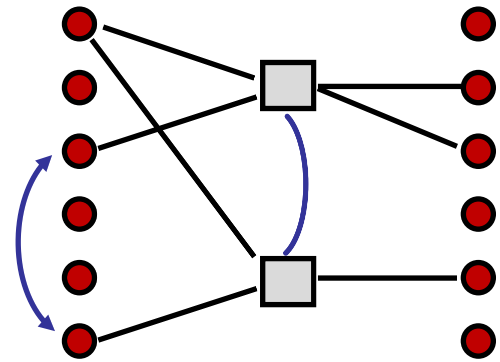


Interference

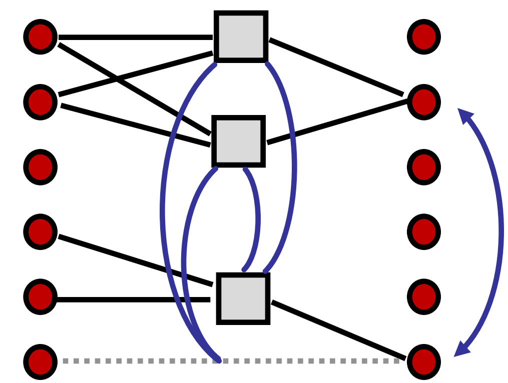
Slide based on Brafman which in turn is based on Ambite, Blyth, and Weld

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



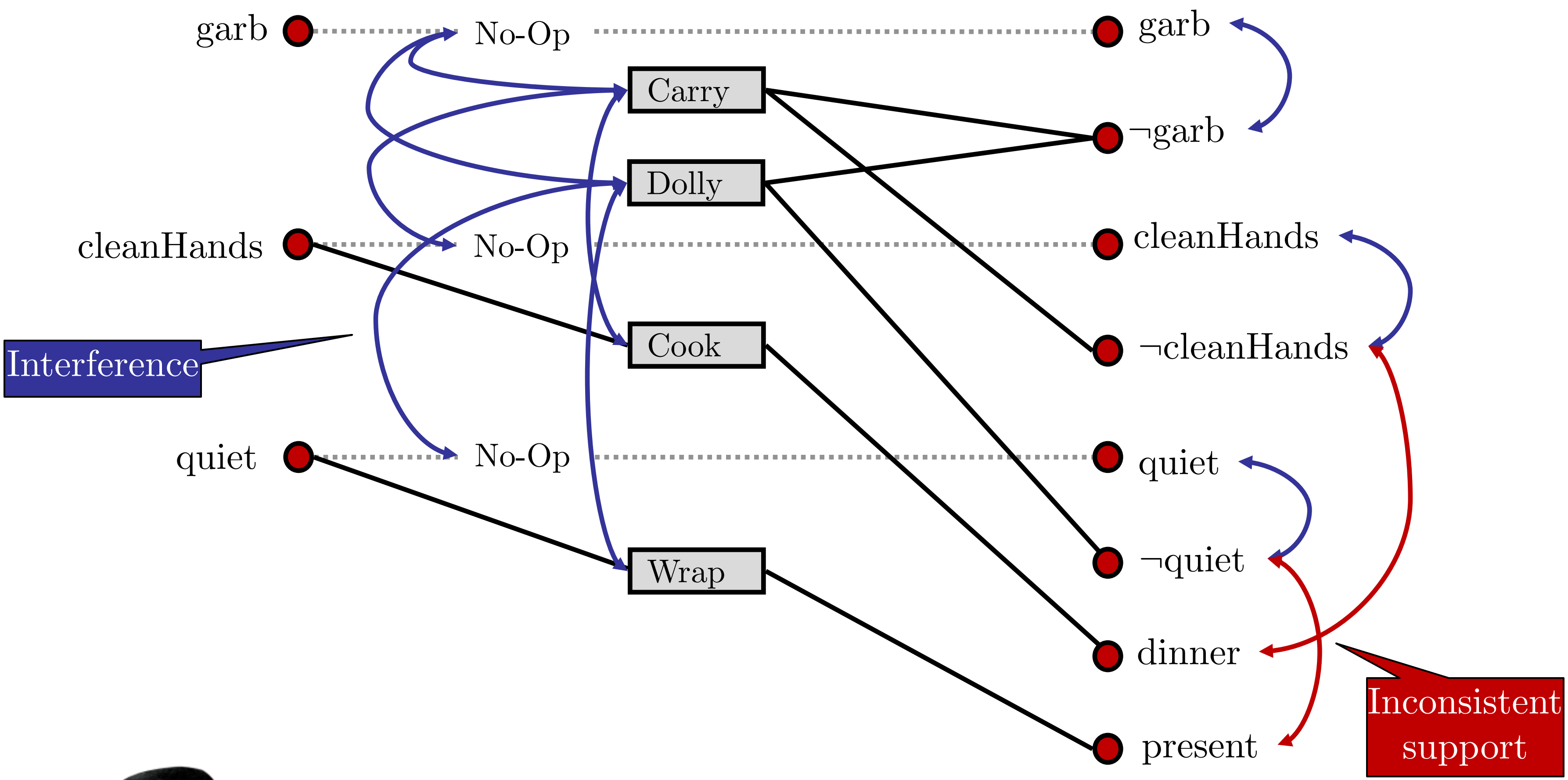
Inconsistent Support

Dinner date example

- Initial state: garbage ∧ cleanHands ∧ quiet
- Goals: dinner ∧ present ∧ ¬garbage
- Actions:
  - Cook: cleanHands ⇒ dinner
  - Wrap: quiet ⇒ present
  - Carry: none ⇒ ¬garbage ∧ ¬ cleanHands
  - Dolly: none ⇒ ¬garbage ∧ ¬ quiet
- What's the plan?

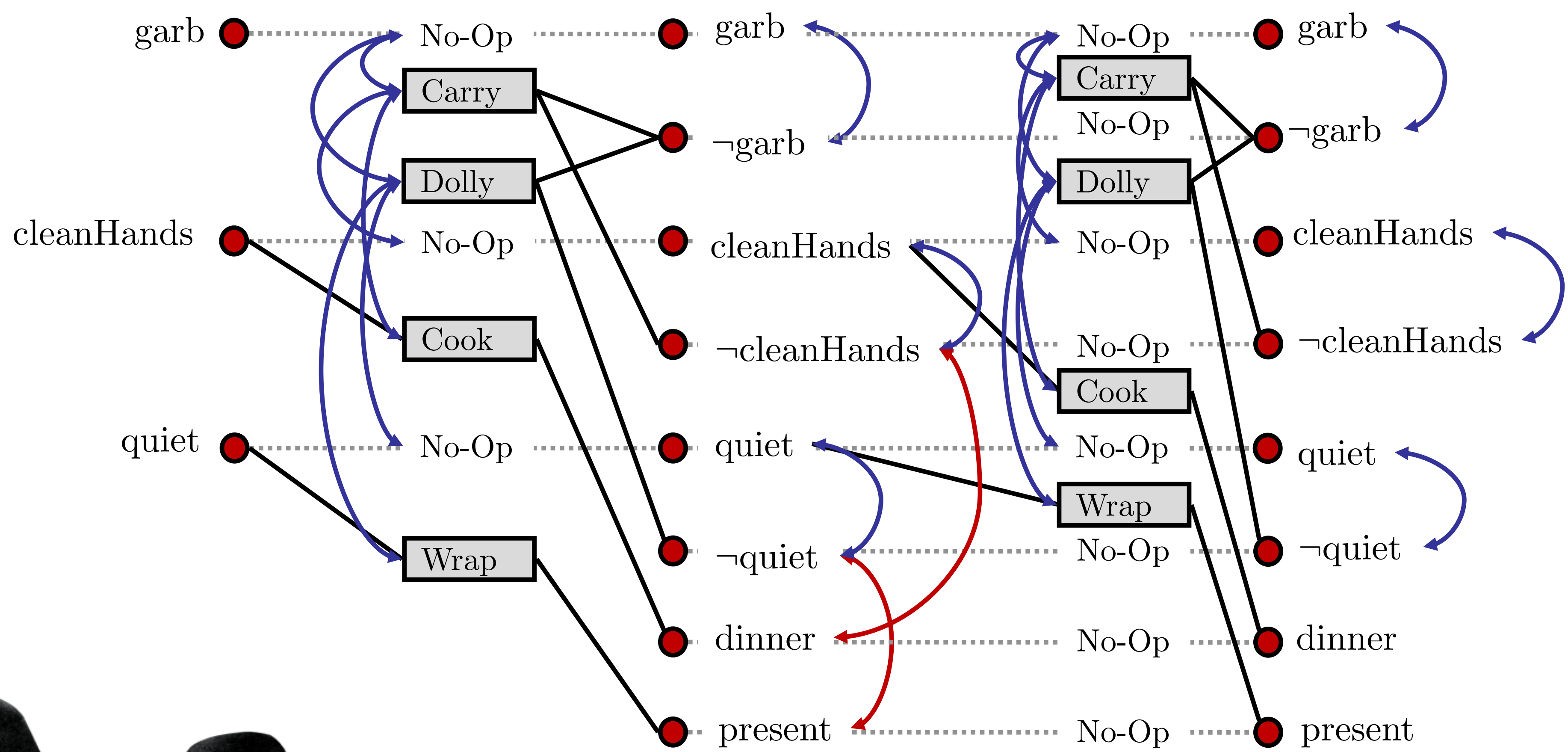
Slide based on Brafman which in turn is based on Ambite, Blyth, and Weld

Dinner date example



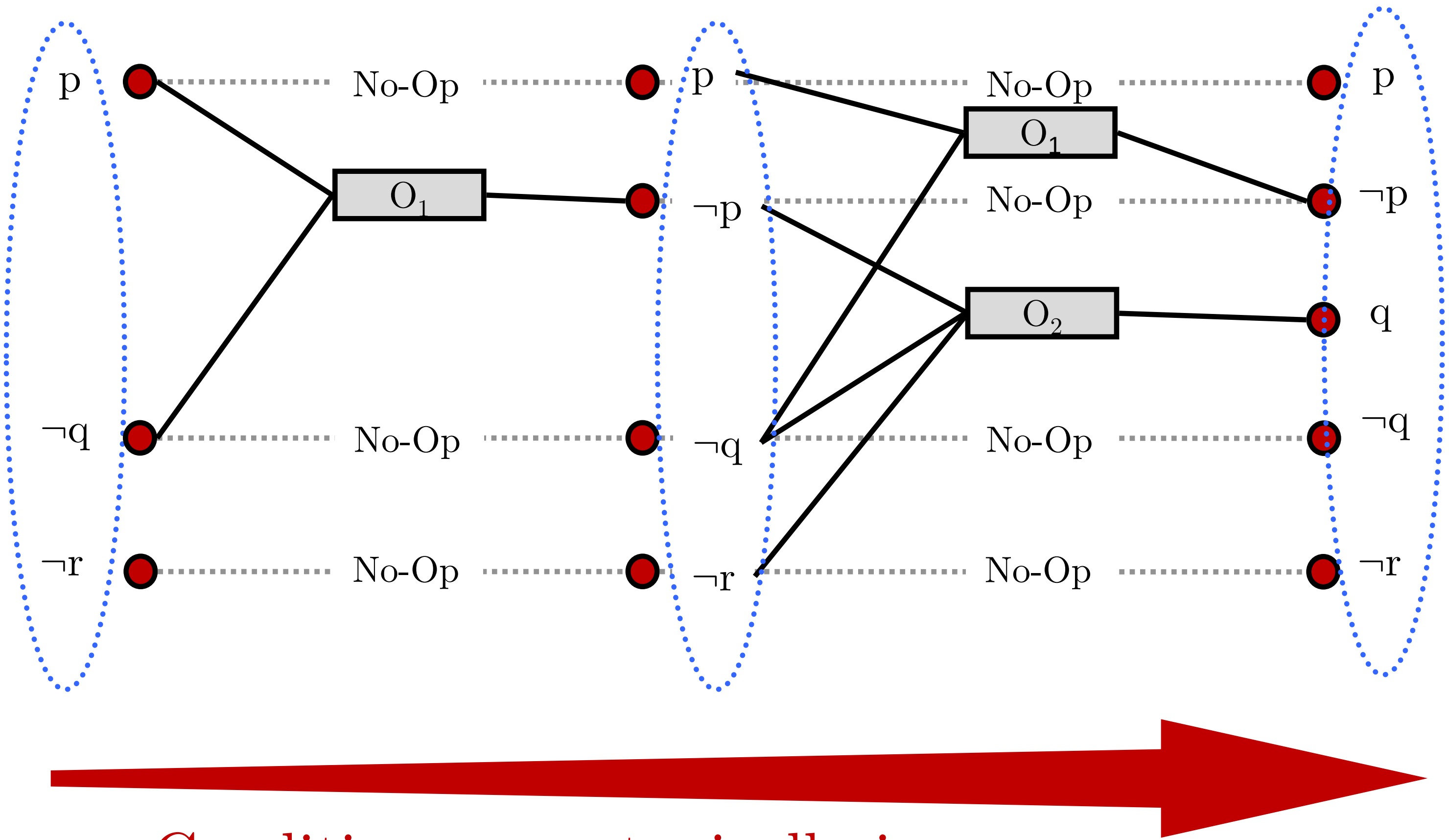
Slide based on Brafman which in turn is based on Ambite, Blyth, and Weld

Dinner date example



Slide based on Brafman which in turn is based on Ambite, Blyth, and Weld

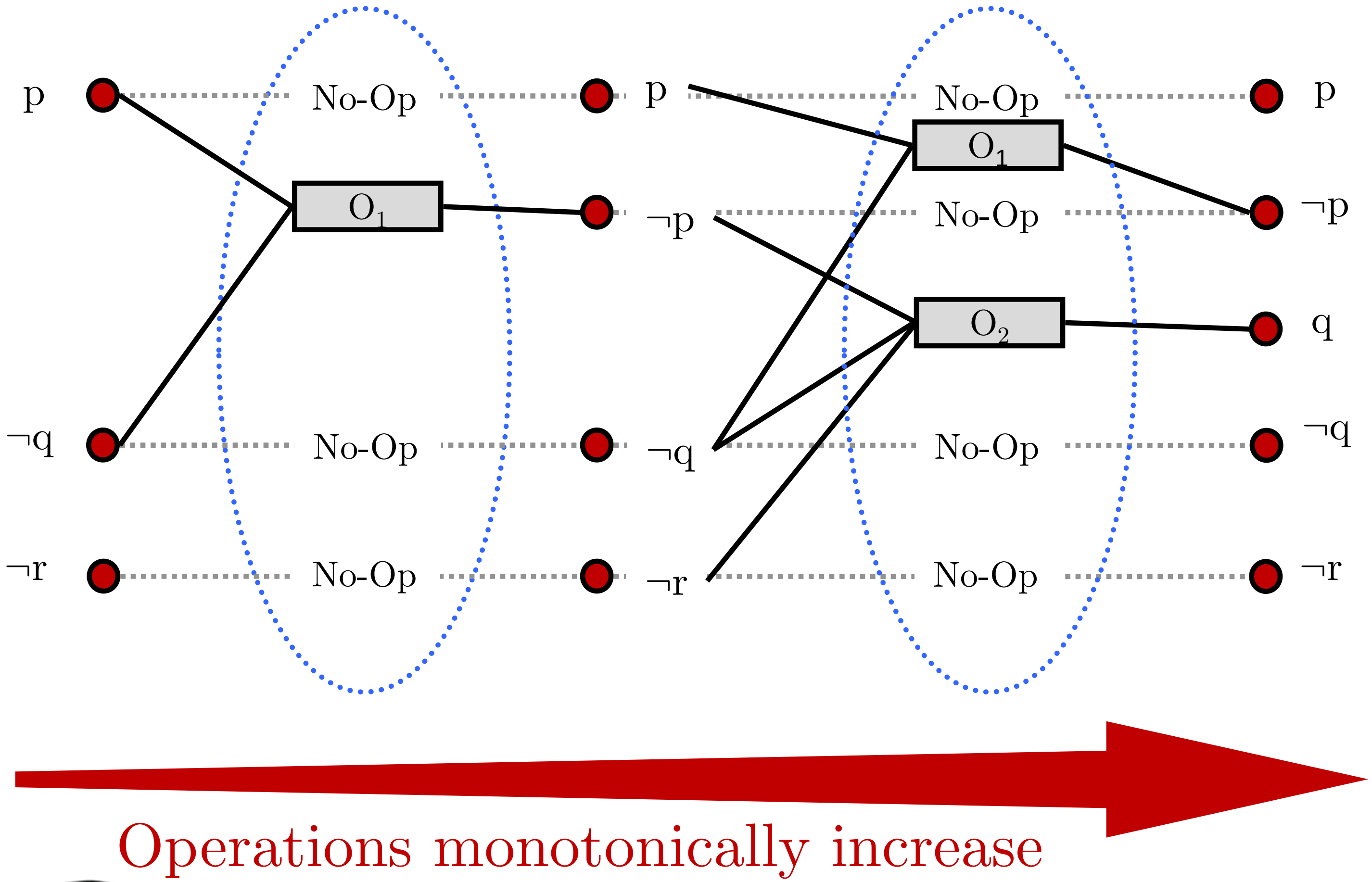
Observation 1



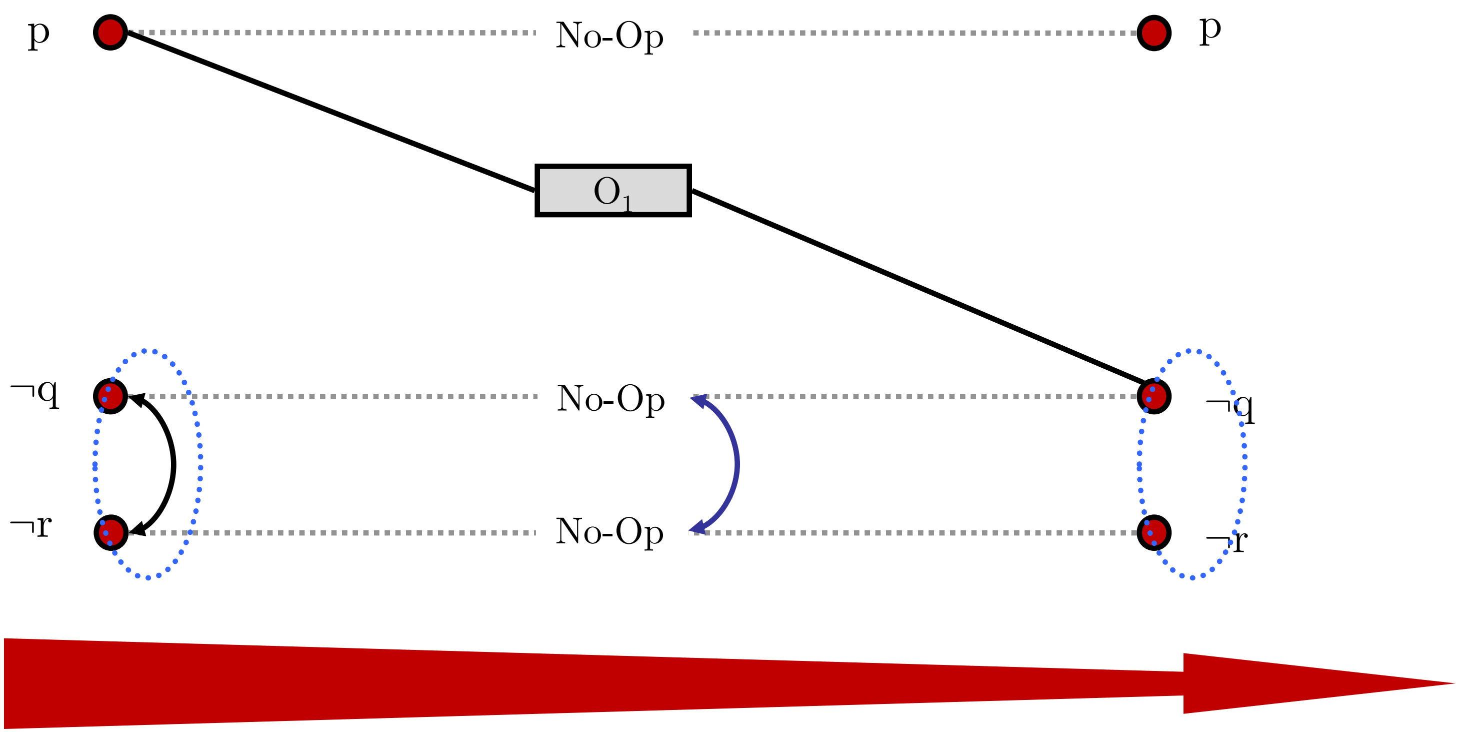
Conditions monotonically increase
(always carried forward by no-ops)

Slide based on Brafman which in turn is based on Ambite, Blyth, and Weld

Observation 2



Observation 3



Proposition mutex relationships monotonically decrease

Observation 4

- Operation mutexes monotonically decrease
- Inconsistent effects and interference are properties of the operations themselves ⇒ hold at every level
- Competing needs: proposition 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 (admissible?)
  - Level sum: sum of level costs (admissible?)
  - Set level: level at which all goals appear without any pair being mutex (admissible?)

The Graphplan algorithm

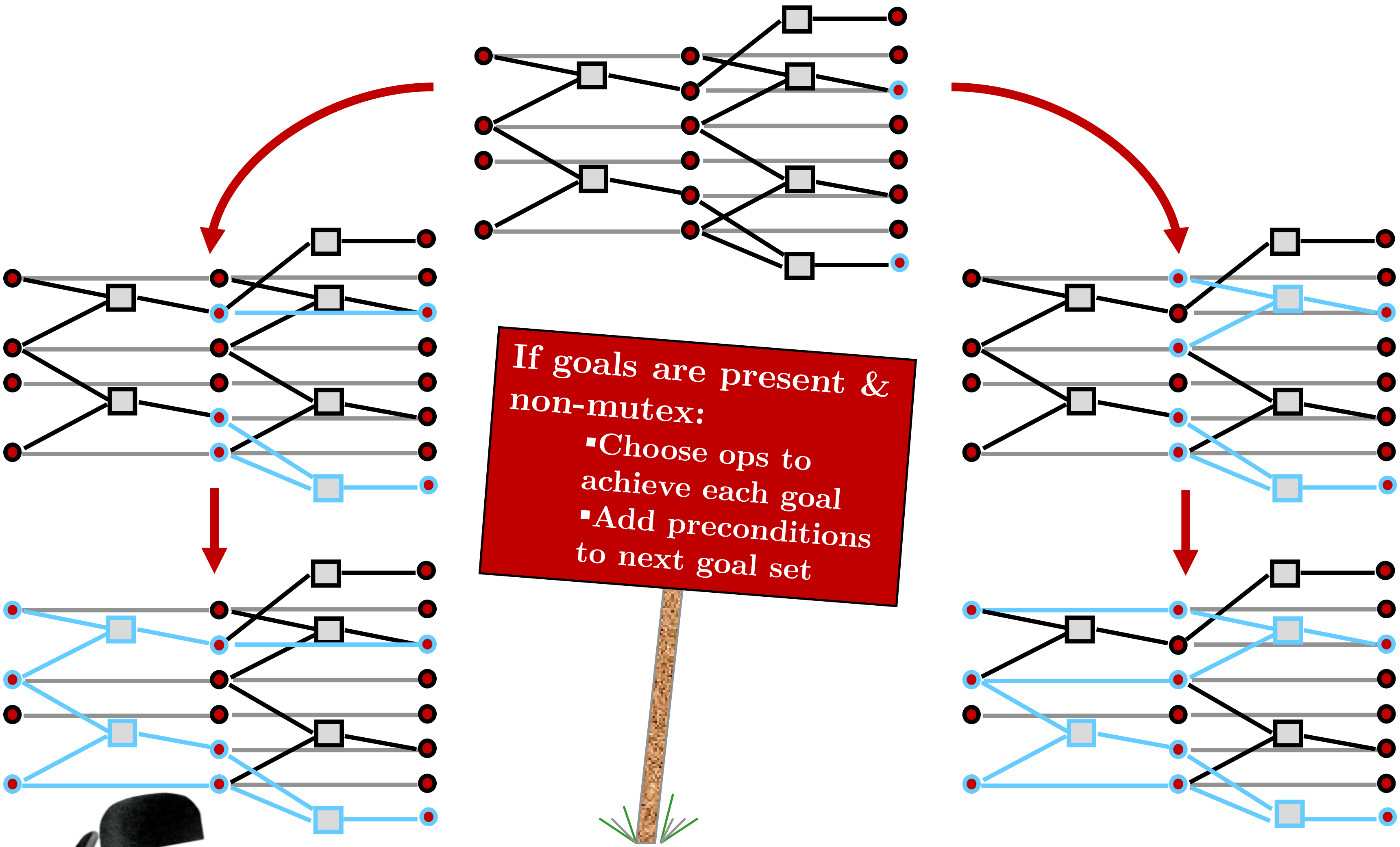
1. Grow the planning graph until all goals are reachable and not mutex
   (If planning graph levels off first, fail)
2. Call Extract-Solution on current planning graph
3. If none found, add a level to the planning graph and try again

Slide based on Brafman which in turn is based on Ambite, Blyth, and Weld

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 which in turn is based on Ambite, Blyth, and Weld

Graphplan guarantees

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