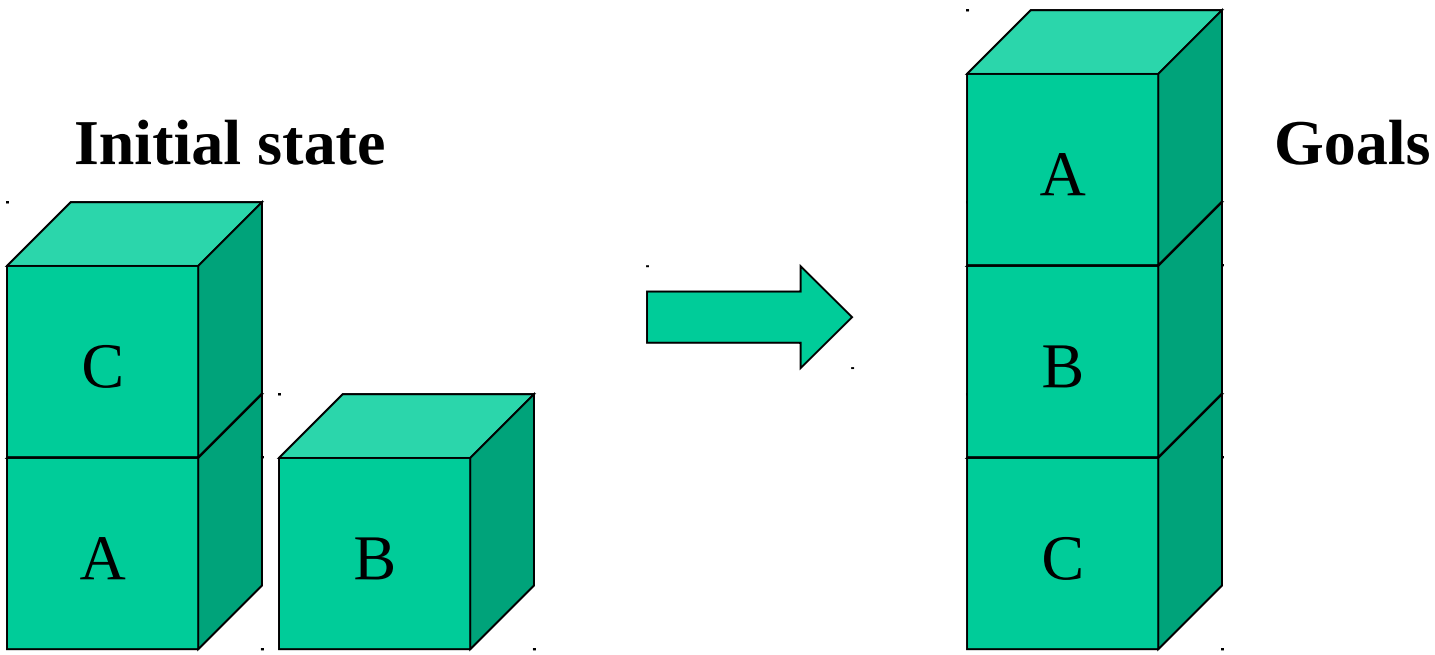
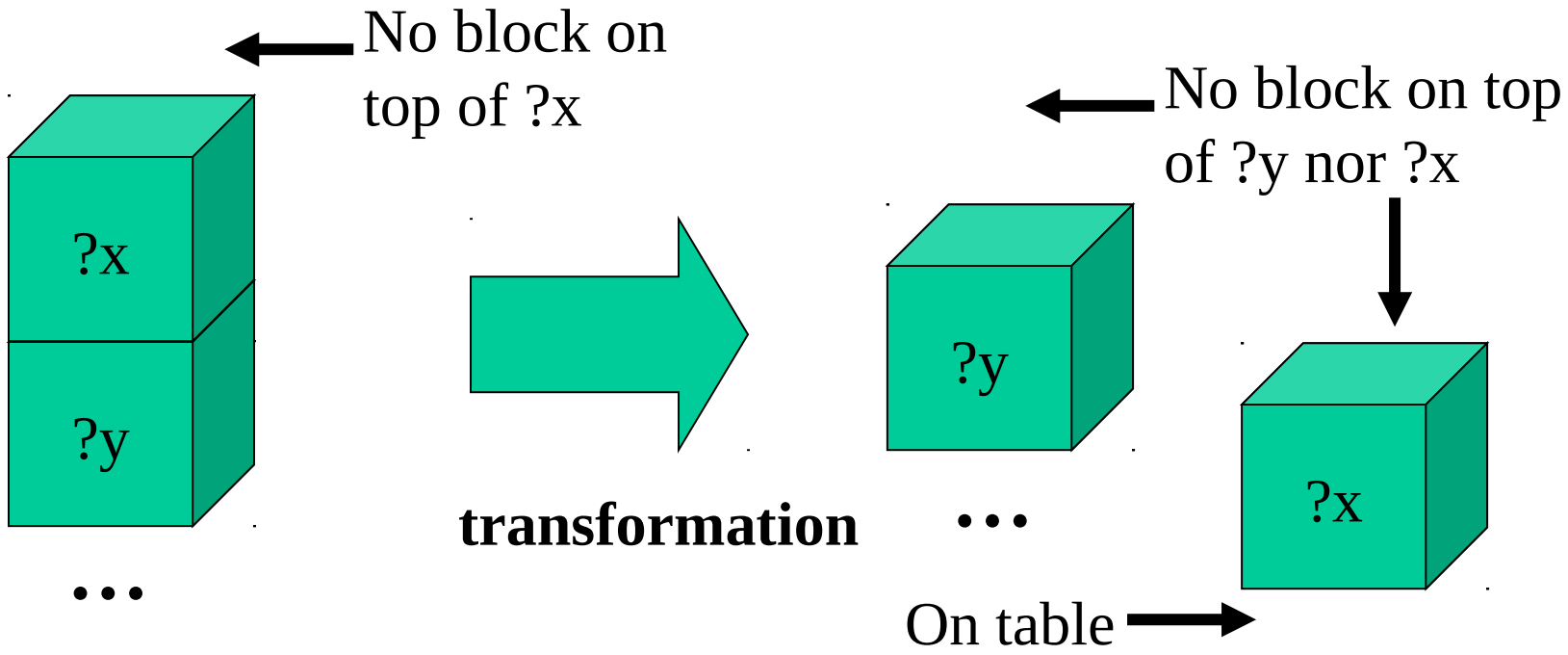
(Classical)
AI Planning

General-Purpose Planning: State & Goals



- **Initial state**: (on A Table) (on C A) (on B Table) (clear B) (clear C)
- **Goals**: (on C Table) (on B C) (on A B) (clear A)

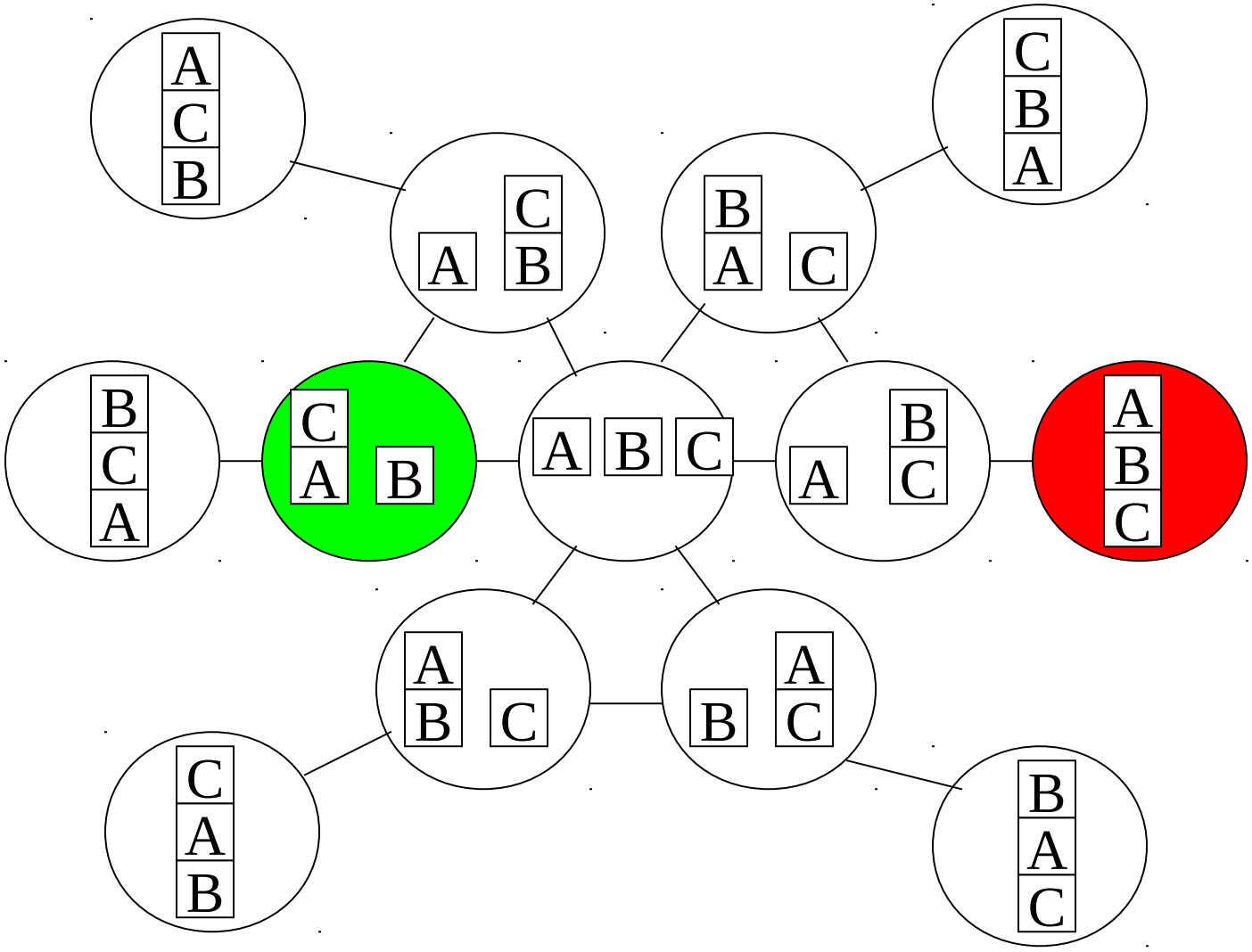
General-Purpose Planning: Operators



Operator: (Unstack ?x)

- **Preconditions**: (on ?x ?y) (clear ?x)
- **Effects**:
  - **Add**: (on ?x table) (clear ?y)
  - **Delete**: (on ?x ?y)

Planning: Search Space



(Michael Moll)

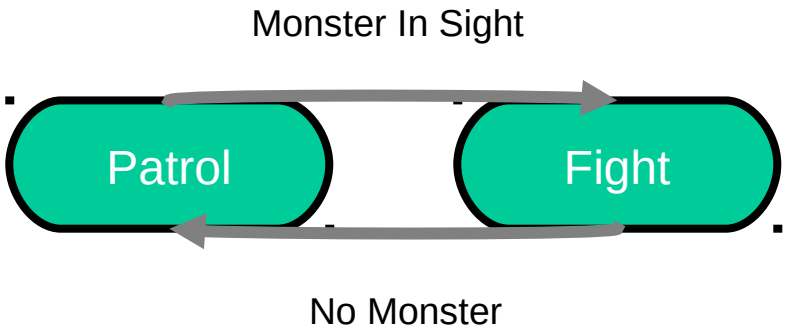
Some Examples

Applications which can be modeled as AI planning problems?

- **Route search**: Find a route between University and the Research Laboratory
- **Project management**: Construct a project plan for organizing an event
- **Military operations**: Develop an air campaign
- **Information gathering**: Find and reserve an airline ticket to travel from source to destination
- **Game playing**: plan the behavior of a computer controlled player
- **Resources control**: Plan the stops of several of elevators in a skyscraper building.

FSM vs AI Planning

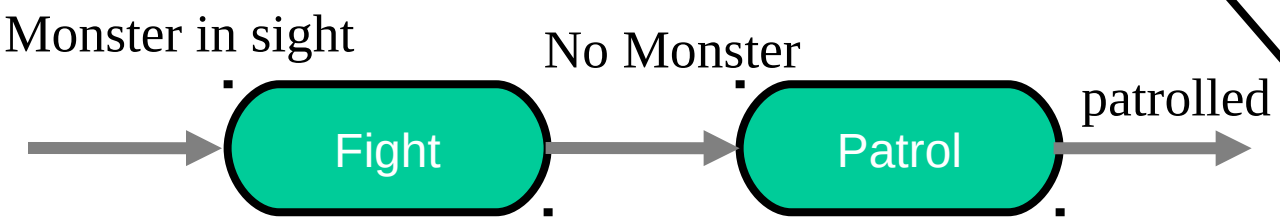
FSM:



Planning Operators

- **Patrol**
  - Preconditions: No Monster
  - Effects: patrolled
- **Fight**
  - Preconditions: Monster in sight
  - Effects: No Monster

A resulting plan:



Neither is more powerful than the other one

But Planning Gives More Flexibility

- "Separates implementation from data" --- Orkin

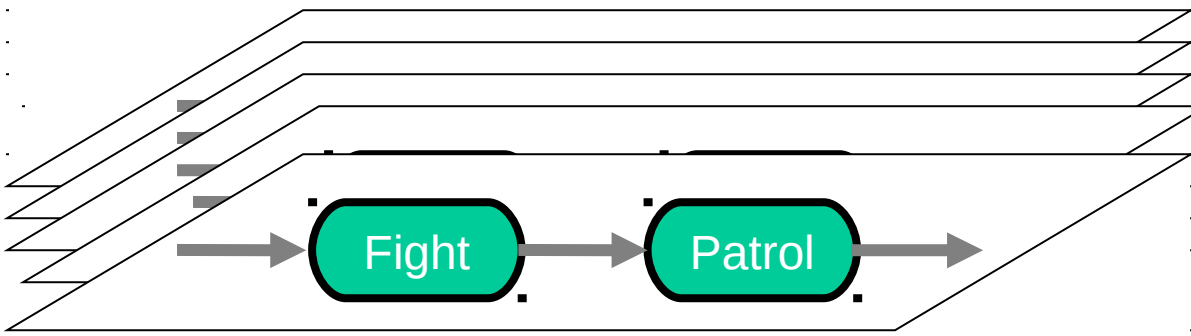
reasoning knowledge

Planning Operators

- **Patrol**
  - Preconditions: No Monster
  - Effects: patrolled
- **Fight**
  - Preconditions: Monster in sight
  - Effects: No Monster

…

Many potential plans:



…

If conditions in the state change making the current plan unfeasible: replan!

But… Does Classical Planning Work for Games?

General Purpose vs. Domain-Specific

Planning: find a sequence of actions to achieve a goal

General purpose: symbolic descriptions of the problems and the domain. The plan generation algorithm the same

Advantage: - opportunity to have clear semantics

Disadvantage: - symbolic description requirement

Domain Specific: The plan generation algorithm depends on the particular domain

Advantage: - can be very efficient

Disadvantage: - lack of clear semantics

- knowledge-engineering for plan generation

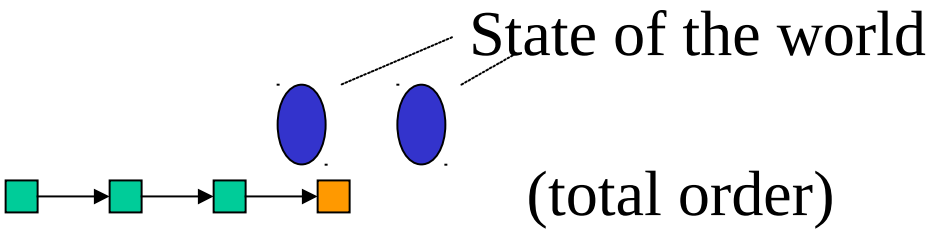
Classes of General-Purpose Planners

General purpose planners can be classified according to the space where the search is performed:

- state
- plan
- Hierarchical

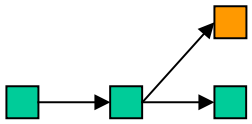
State- and Plan-Space Planning

- **State-space** planners transform the state of the world. These planners search for a sequence of transformations linking the starting state and a final state



(total order)

- **Plan-space** planners transform the plans. These planners search for a a plan satisfying certain conditions

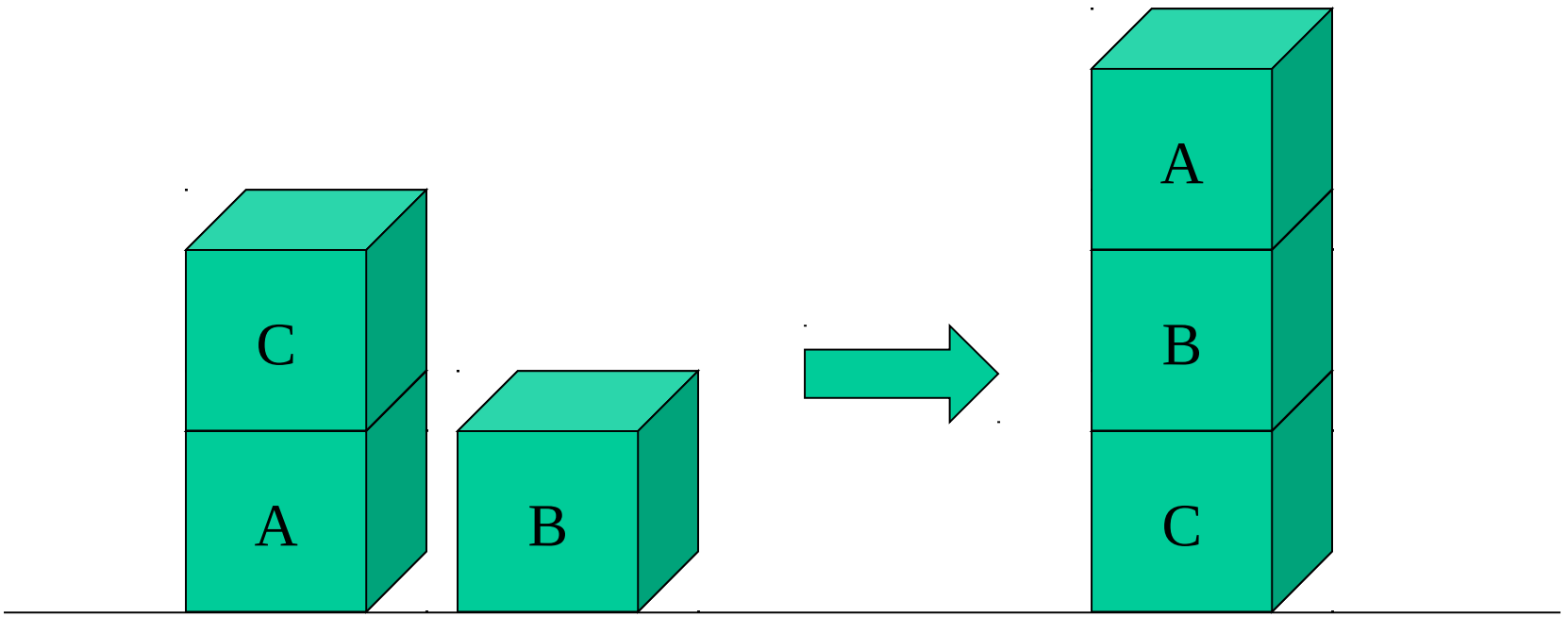


(partial-order, least-commitment)

Why Plan-Space Planning?

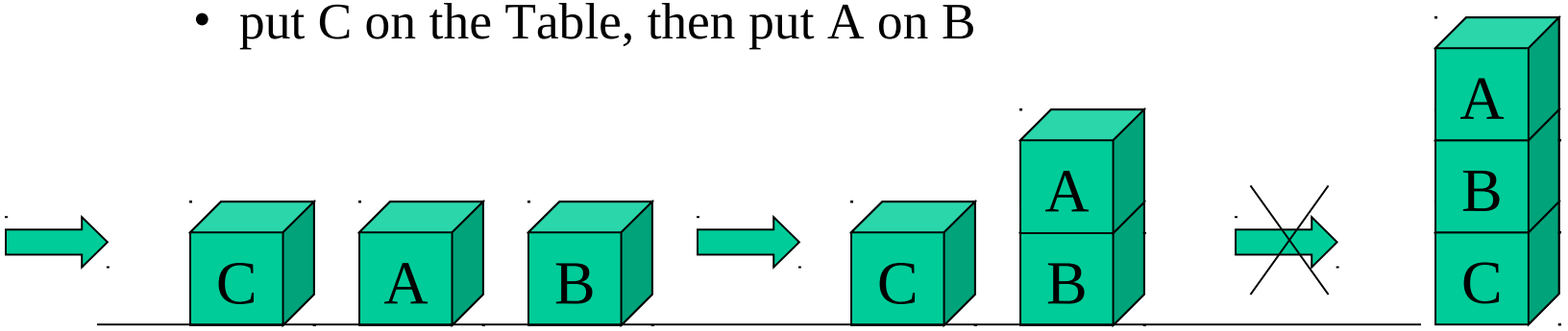
- 1. Motivation: “Sussman Anomaly”
  - Two subgoals to achieve:

    (on A B) (on B C)



Why Plan-Space Planning?

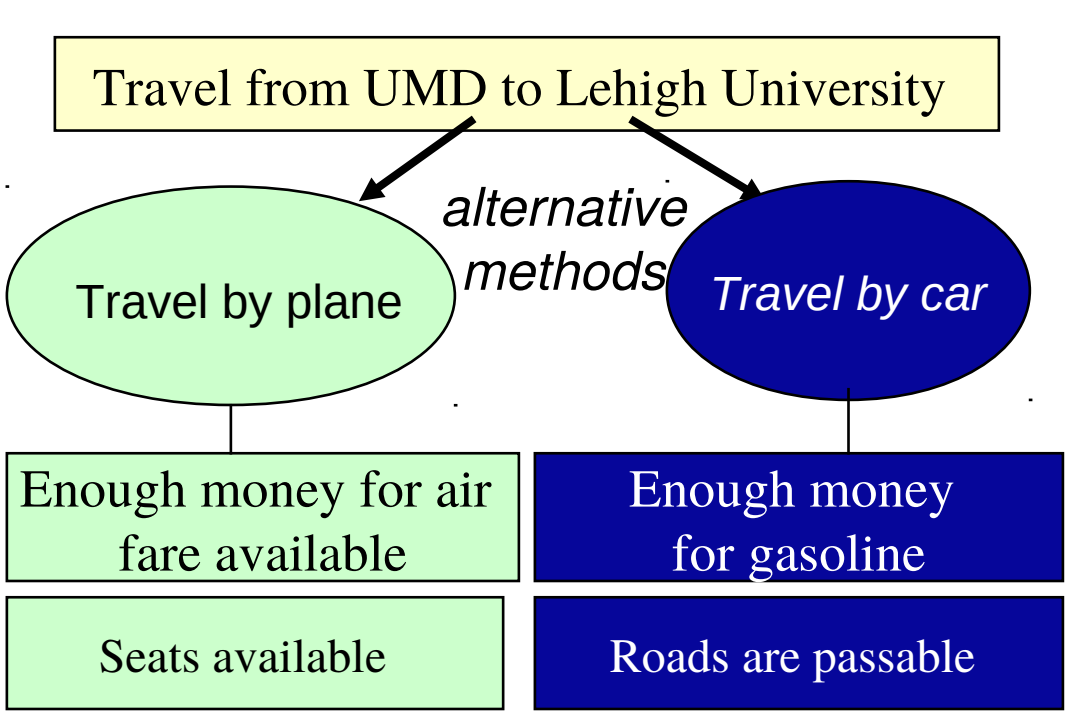
- Problem of state-space search:
  - Try (on A B) first:
    - put C on the Table, then put A on B

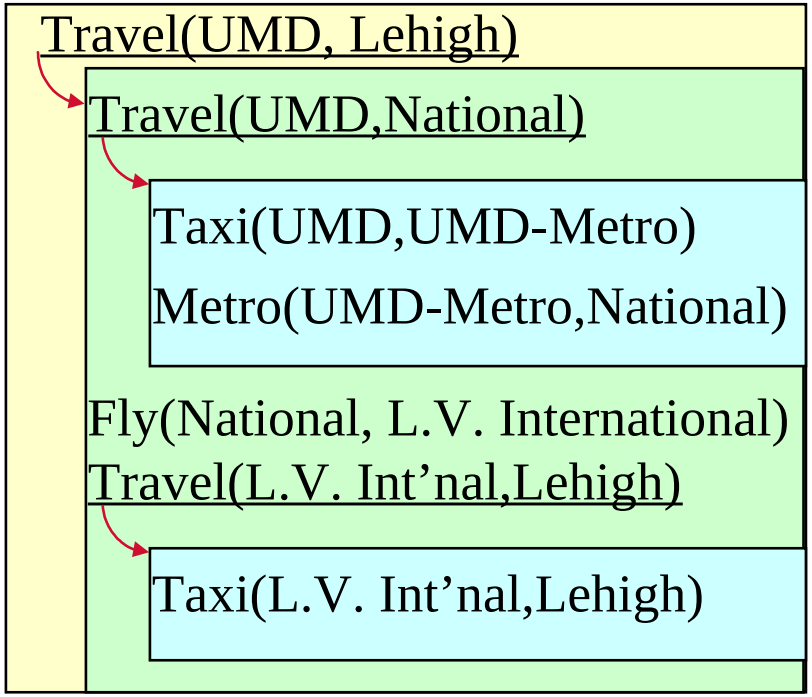


- Accidentally wind up with A on B when B is still on the Table
- We can not get B on C without taking A off B
- Try to solve the first subgoal first appears to be mistaken

Hierarchical (HTN) Planning

Principle: Complex tasks are decomposed into simpler tasks. The goal is to decompose all the tasks into *primitive* tasks, which define actions that change the world.





Application to Computer Bridge

- Chess: better than all but the best humans
- Bridge: worse than many good players

- Why bridge is difficult for computers
  - It is an imperfect information game
  - Don't know what cards the others have (except the dummy)
  - Many possible card distributions, so many possible moves
- If we encode the additional moves as additional branches in the game tree, this increases the number of nodes exponentially
  - worst case: about 6×10^{44} leaf nodes
  - average case: about 10^{24} leaf nodes

Not enough time to search the game tree

How to Reduce the Size of the Game Tree?

- Bridge is a game of planning
  - Declarer plans how to play the hand by combining various strategies (ruffing, finessing, etc.)
  - If a move doesn't fit into a sensible strategy, then it probably doesn't need to be considered
- HTN approach for declarer play
  - Use HTN planning to generate a game tree in which each move corresponds to a different *strategy*, not a different *card*
    - Reduces average game-tree size to about 26,000 leaf nodes
- Bridge Baron: implements HTN planning
  - Won the 1997 World Bridge Computer Challenge
  - All commercial versions of Bridge Baron since 1997 have include an HTN planner (has sold many thousands of copies)

Universal Classical Planning (UCP)

(Khambampati, 1997)

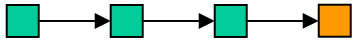
- Loop:
  - If the current *partial plan* is a solution, then exit
  - Nondeterministically choose a way to *refine* the plan

partially instantiated steps, plus constraints

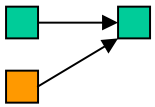
add steps & constraints

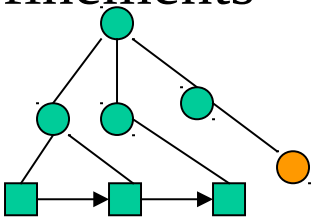
- Some of the possible **refinements**
  - Forward & backward state-space refinement
  - Plan-space refinement
  - Hierarchical refinements

State-space



Plan-space





Abstract Example