Lecture slides for

Automated Planning: Theory and Practice

Chapter 1
Introduction

Dana S. Nau

University of Maryland

Updated 3:03 PM December 14, 2013

Some Dictionary Definitions of "Plan"

plan *n.*

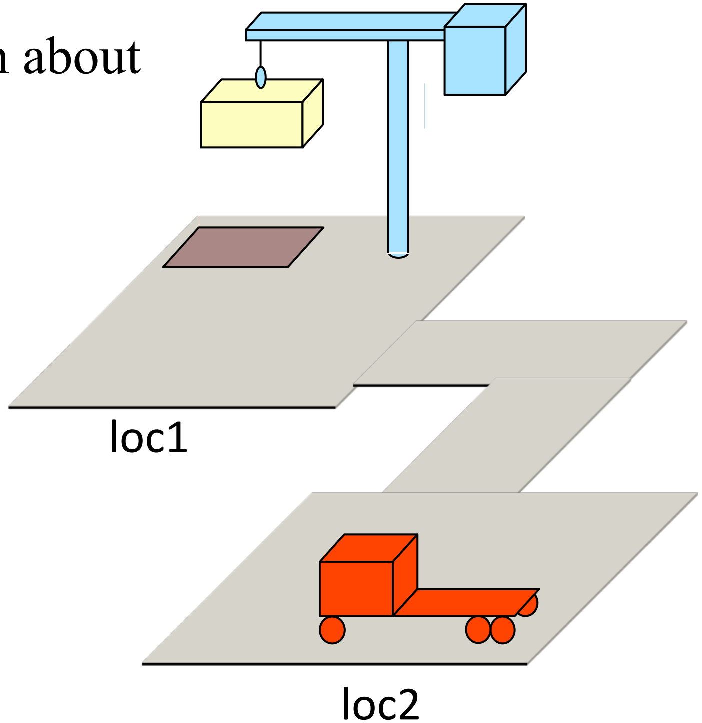
1. A scheme, program, or method worked out beforehand for the accomplishment of an objective: *a plan of attack.*
2. A proposed or tentative project or course of action: *had no plans for the evening.*
3. A systematic arrangement of elements or important parts; a configuration or outline: *a seating plan; the plan of a story.*
4. A drawing or diagram made to scale showing the structure or arrangement of something.
5. A program or policy stipulating a service or benefit: *a pension plan.*

[a representation] of future behavior … usually a set of actions, with temporal and other constraints on them, for execution by some agent or agents.

– Austin Tate, *MIT Encyclopedia of the Cognitive Sciences*, 1999

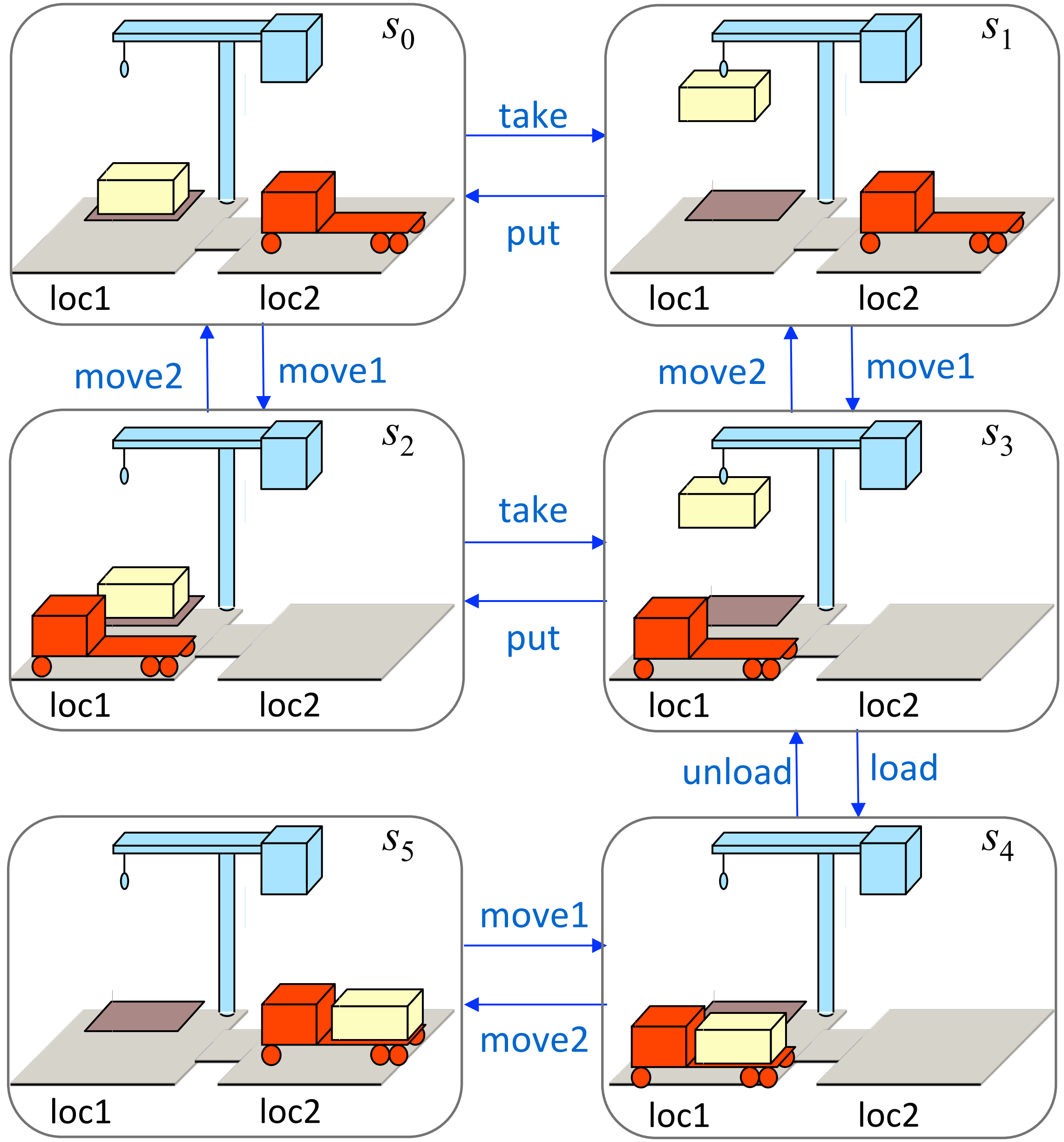
Abstraction

- Real world is absurdly complex, need to approximate
  - Only represent what the planner needs to reason about
- **State transition system $\Sigma = (S, A, E, \gamma)$**
  - S = {abstract states}
    - e.g., states might include a robot's location, but not its position and orientation
  - A = {abstract actions}
    - e.g., "move robot from loc2 to loc1" may need complex lower-level implementation
  - E = {abstract exogenous events}
    - Not under the agent's control
  - γ = state transition function
    - Gives the next state, or possible next states, after an action or event
    - $\gamma: S \times (A \cup E) \rightarrow S$ or $\gamma: S \times (A \cup E) \rightarrow 2^S$
- In some cases, avoid ambiguity by writing S_Σ, A_Σ, E_Σ, γ_Σ



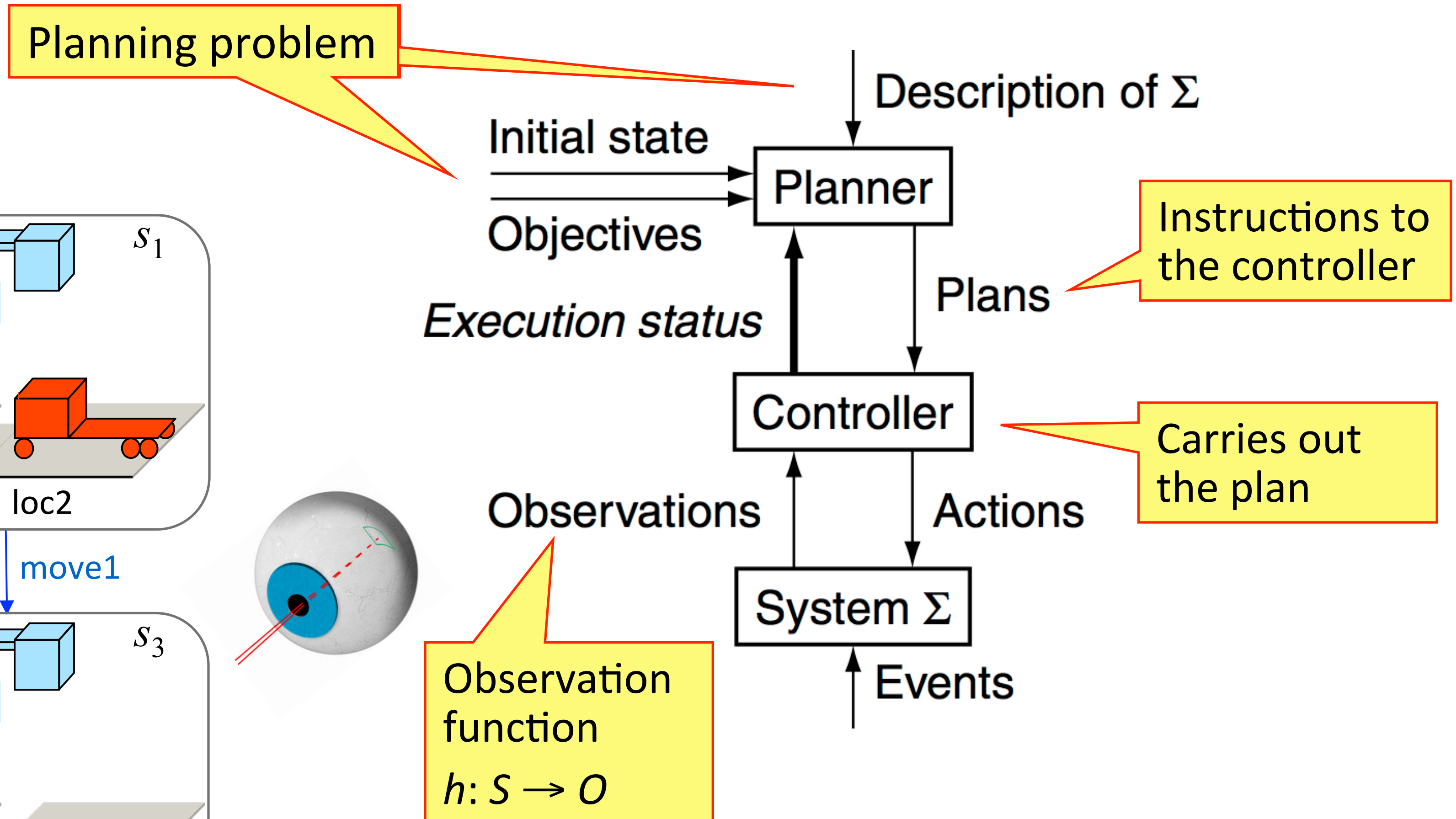
State Transition System

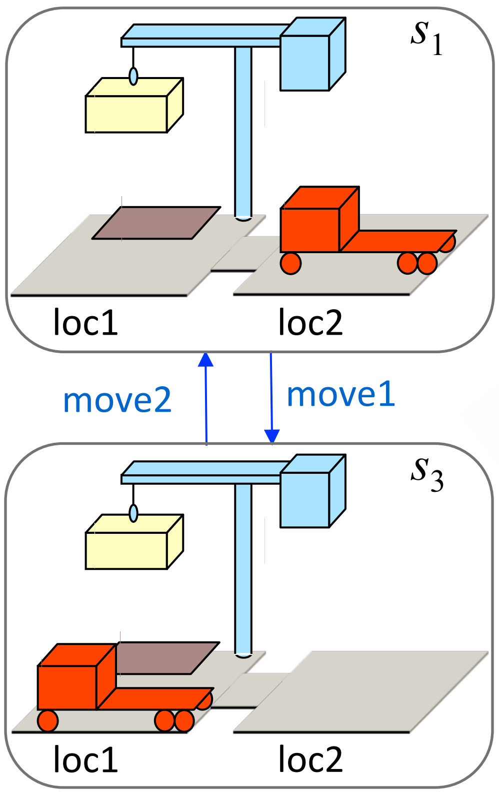
- $\Sigma = (S,A,E,\gamma)$
  - S = {states}
  - A = {actions}
  - E = {exogenous events}
  - γ = state-transition func.
- Example:
  - $S = \{s_0, \ldots, s_5\}$
  - A = {move1, move2, put, take, load, unload}
  - $E = \{\}$
    - so write $\Sigma = (S,A,\gamma)$
  - $\gamma: S \times A \rightarrow S$
    - see the arrows



Dock Worker Robots (DWR) example

Conceptual Model



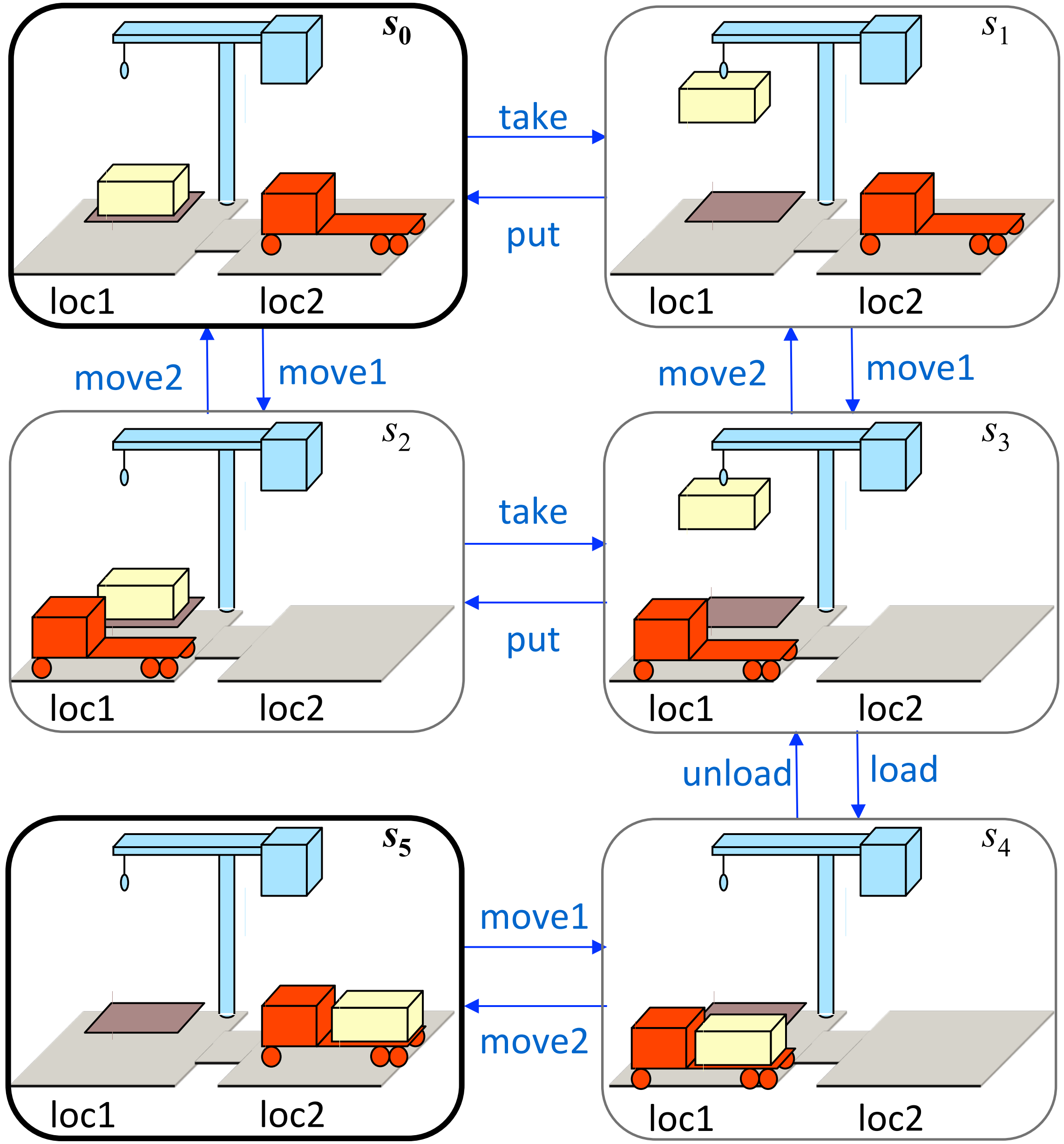


- Control may involve lower-level planning and/or plan execution
  - e.g., how to move from one location to another

Planning Problem

- Description of Σ
- Initial state or set of states
- Objective
  - Goal state, set of goal states, set of tasks, "trajectory" of states, objective function, …

- e.g.,
  - Initial state = s_0
  - Goal state = s_5



Dock Worker Robots (DWR) example

Plans

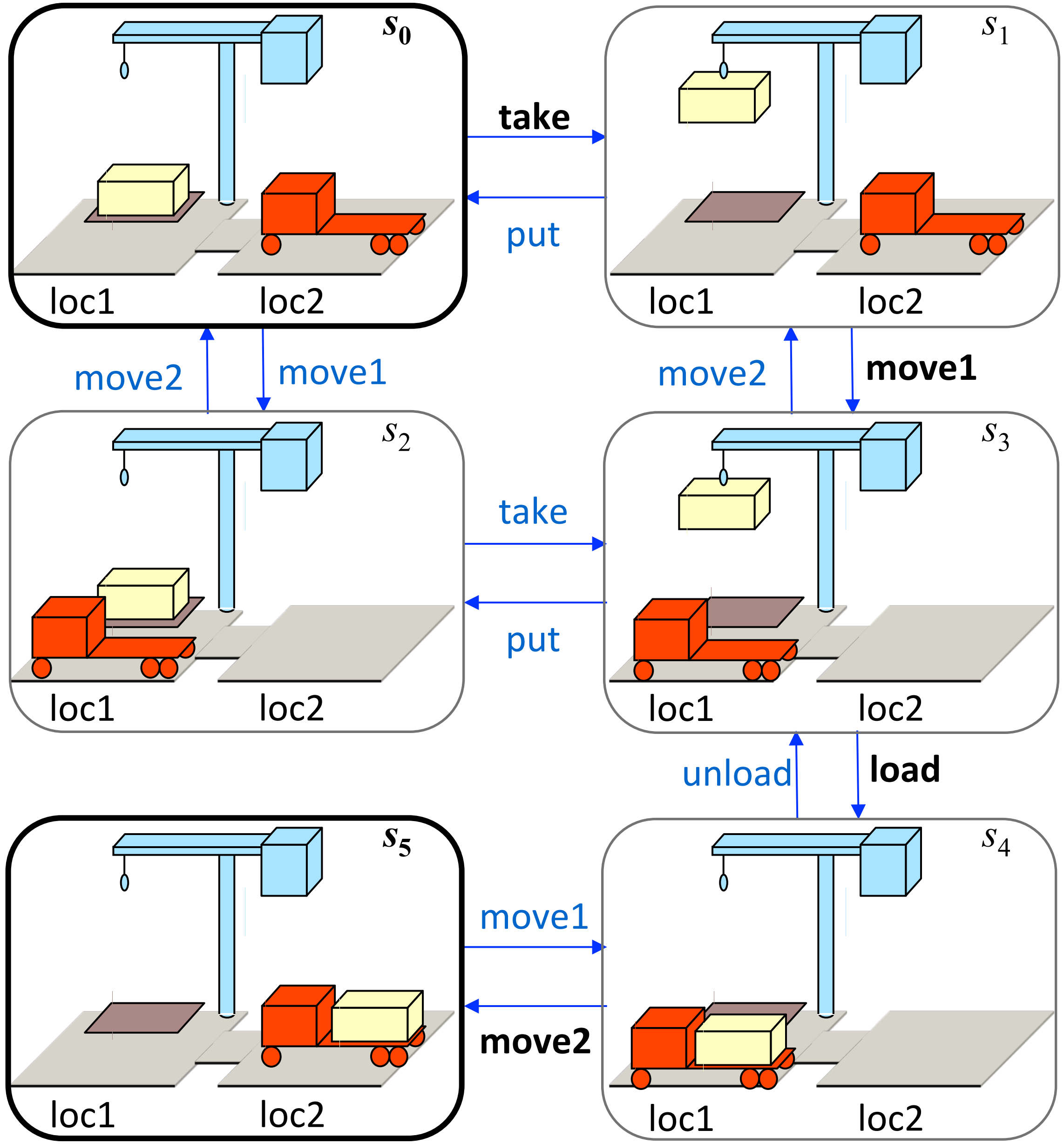
- **Classical plan**: a sequence of actions

  ⟨take, move1, load, move2⟩

- **Policy**: partial function from S into A

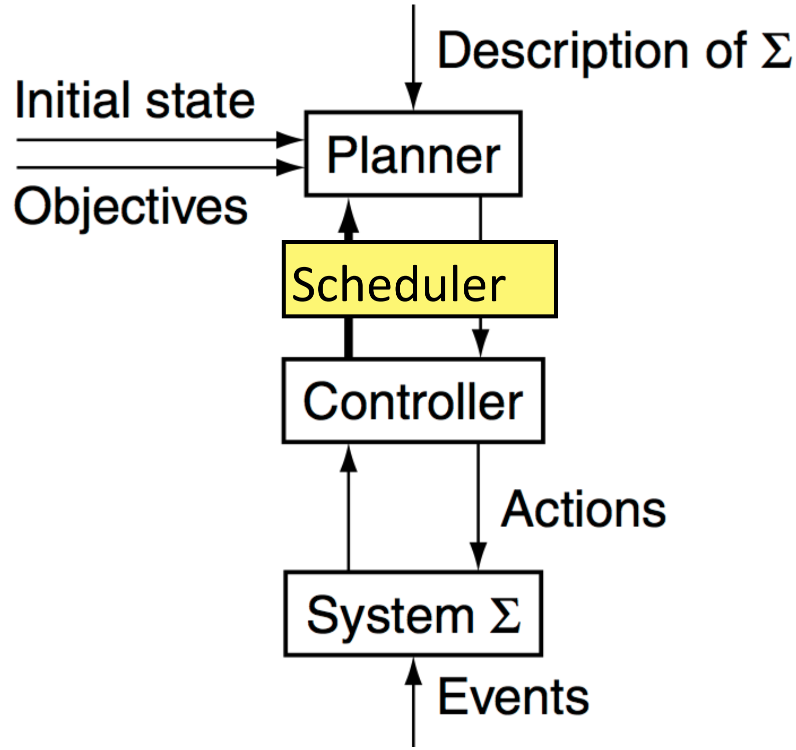
  $\{(s_0$, take),
  $(s_1$, move1),
  $(s_3$, load),
  $(s_4$, move2)$\}$

- Both, if executed starting at s_0, produce s_3



Dock Worker Robots (DWR) example

Planning Versus Scheduling



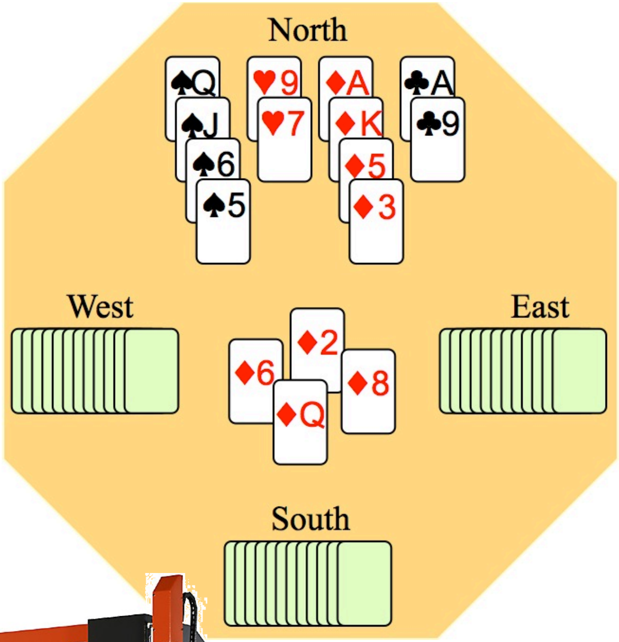
- Scheduling
  - Decide when and how to perform a given set of actions
    - Time constraints
    - Resource constraints
    - Objective functions
  - Typically NP-complete
- Planning
  - Decide what actions to use to achieve some set of objectives
  - Can be much worse than NP-complete; worst case is undecidable
- Scheduling problems may require replanning

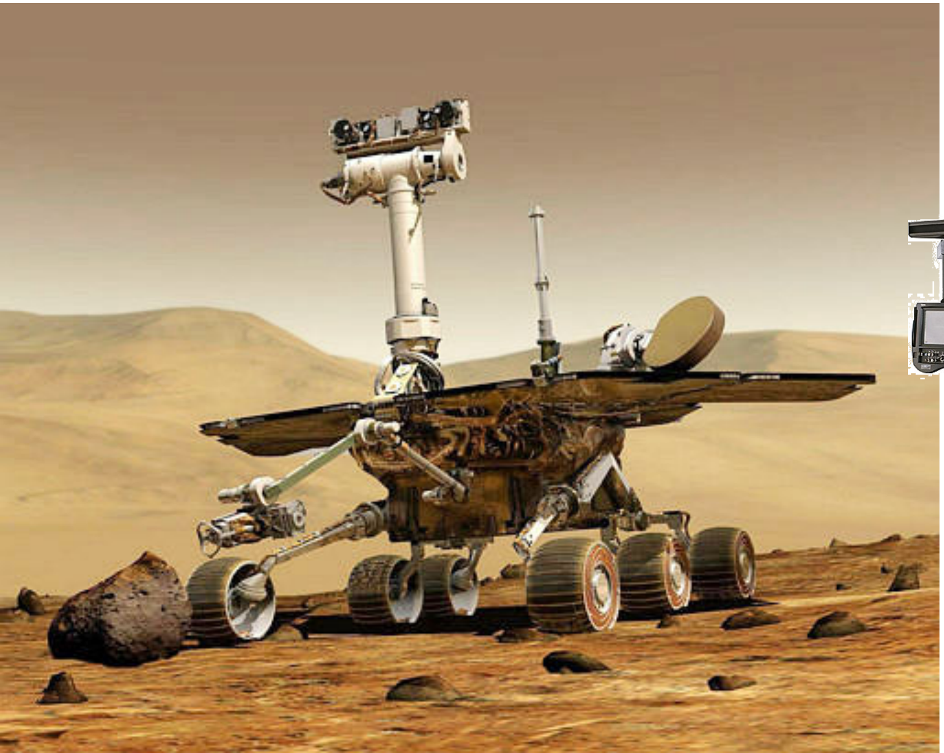
Three Main Types of Planners

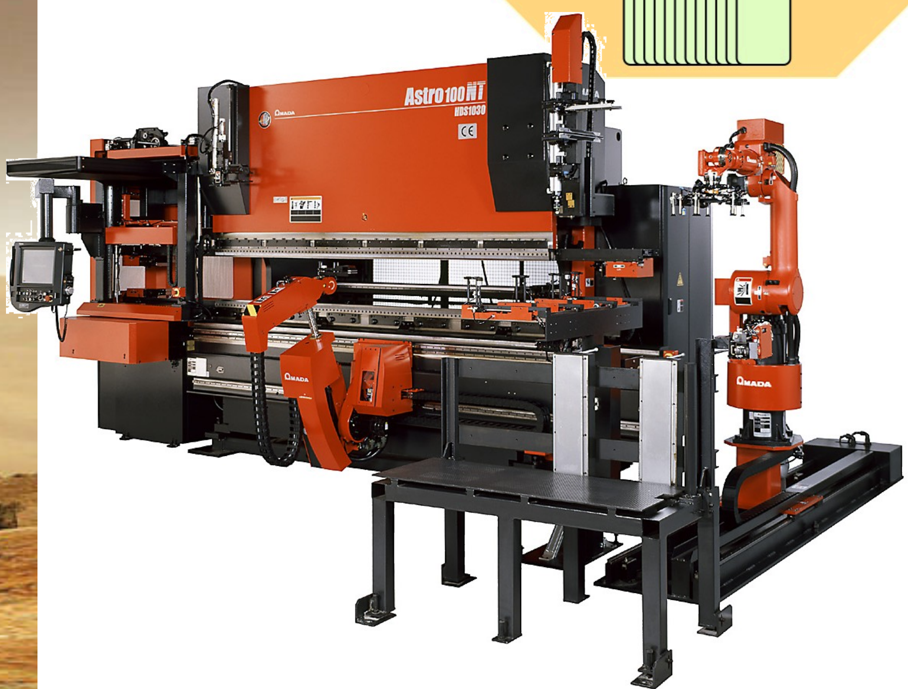
1. Domain-specific
   - Made or tuned for a specific planning domain
   - Won't work well (if at all) in other planning domains
2. Domain-independent
   - In principle, works in any planning domain
   - In practice, need restrictions on what kind of planning domain
3. Configurable
   - Domain-independent planning engine
   - Input includes info about how to solve problems in some domain

1. Domain-Specific Planners (Chapters 19-23)

- Most successful real-world planning systems work this way
  - Mars exploration, sheet-metal bending, playing bridge, etc.
- Often use problem-specific techniques that are difficult to generalize to other planning domains

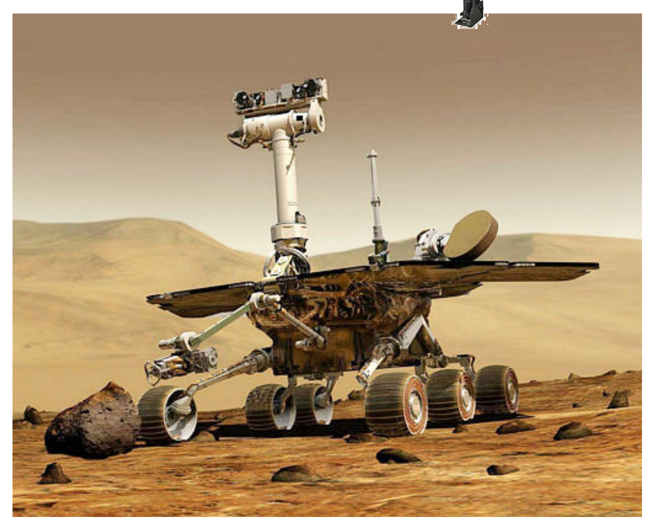






Types of Planners
2. Domain-Independent

- In principle, works in any planning domain
- No domain-specific knowledge except the description of the system Σ
- In practice,
  - Not feasible to make domain-independent planners work well in all possible planning domains



- Make simplifying assumptions to restrict the set of domains
  - *Classical planning*
  - Historical focus of most research on automated planning

Restrictive Assumptions

A0: Finite system:

- finitely many states, actions, events

A1: Fully observable:

- the controller always Σ's current state

A2: Deterministic:

- each action has only one outcome

A3: Static (no exogenous events):

- no changes but the controller's actions

A4: Attainment goals:

- a set of goal states S_g

A5: Sequential plans:

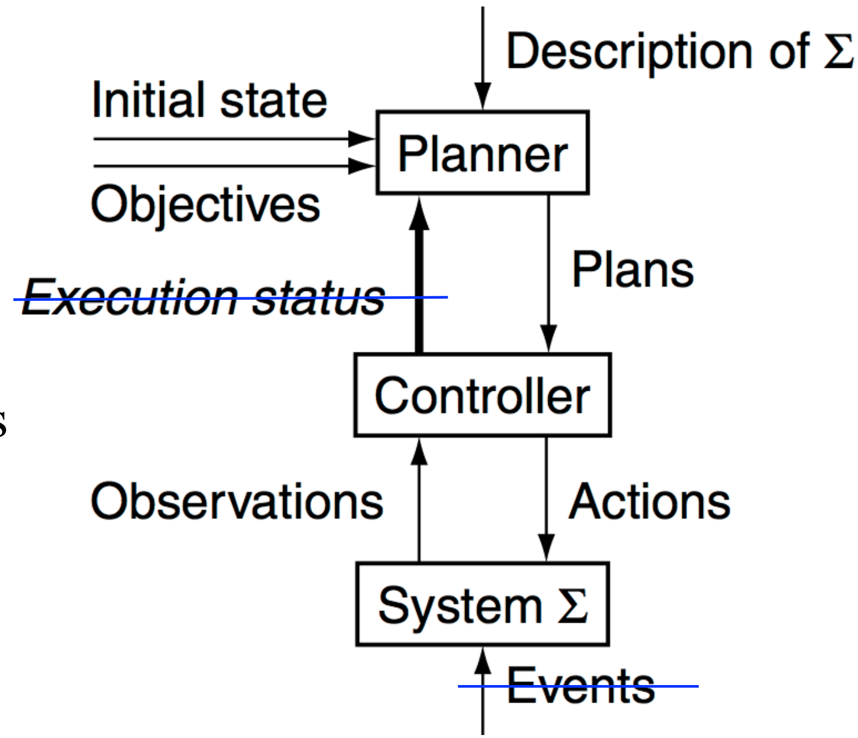
- a plan is a linearly ordered sequence of actions $(a_1, a_2, \ldots a_n)$

A6: Implicit time:

- no time durations; linear sequence of instantaneous states

A7: Off-line planning:

- planner doesn't know the execution status



Classical Planning (Chapters 2–9)

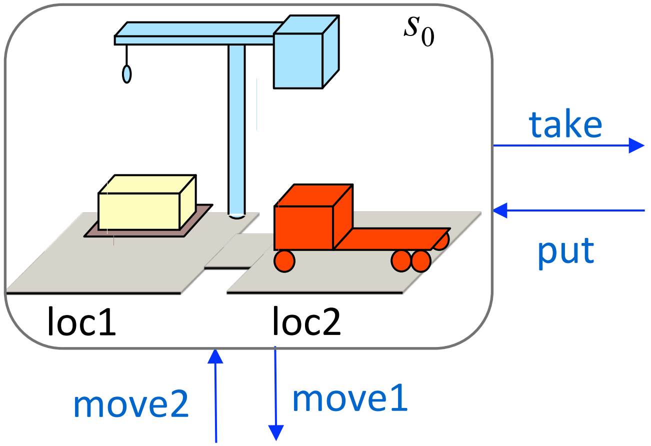
- Classical planning requires all eight restrictive assumptions
  - Offline generation of action sequences for a deterministic, static, finite system, with complete knowledge, attainment goals, and implicit time
- Reduces to the following problem:
  - Given a planning problem $\mathcal{P} = (\Sigma, s_0, S_g)$
  - Find a sequence of actions $(a_1, a_2, \ldots a_n)$ that produces a sequence of state transitions $(s_1, s_2, \ldots, s_n)$ such that s_n is in S_g.
- This is just path-searching in a graph
  - Nodes = states
  - Edges = actions
- Is this trivial?

Classical Planning (Chapters 2–9)

- Generalize the earlier example:

  5 locations,
  3 robot vehicles,
  100 containers,
  3 pallets to stack containers on

  - Then there are 10^{277} states



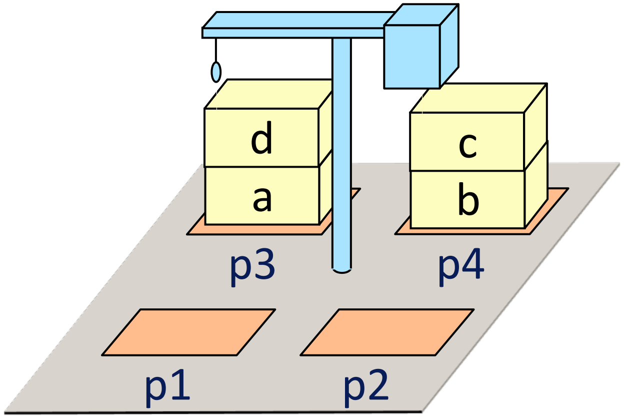
- Number of particles in the universe is only about 10^{87}
  - The example is more than 10^{190} times as large

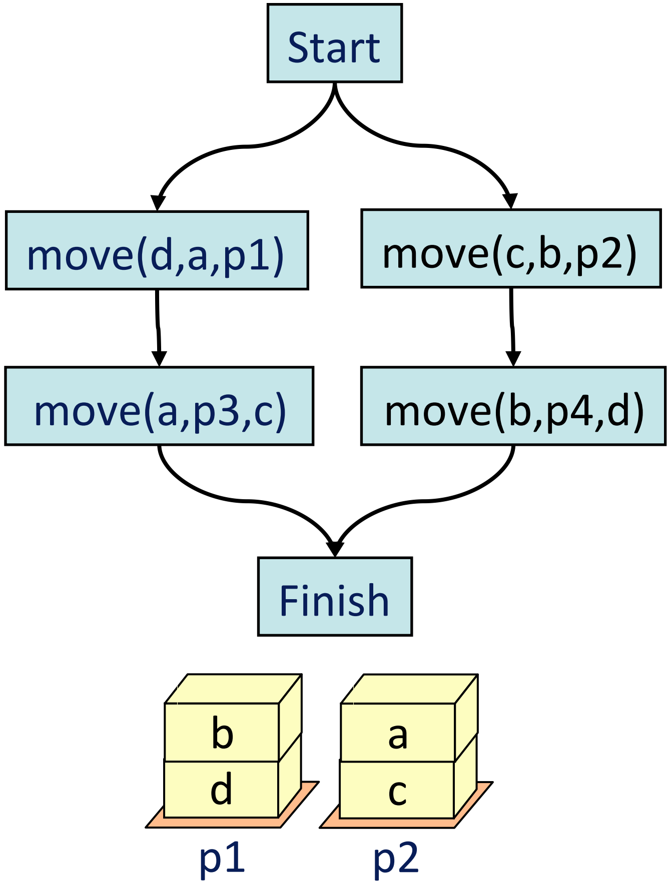
- Automated-planning research has been heavily dominated by classical planning
  - Dozens (hundreds?) of different algorithms

Plan-Space Planning (Chapter 5)

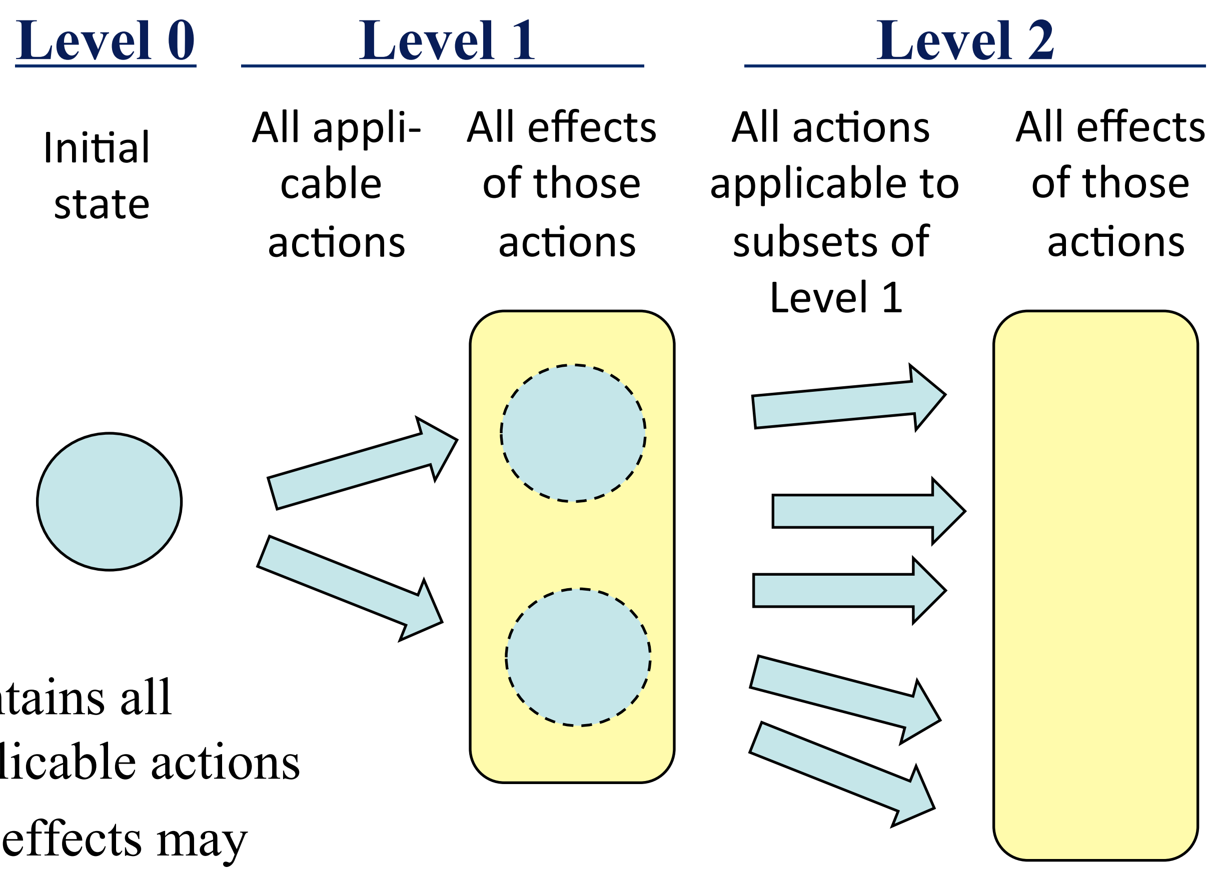
- Decompose sets of goals into the individual goals
- Plan for them separately
  - Bookkeeping info to detect and resolve interactions
- Produce a partially ordered plan that retains as much flexibility as possible

- The Mars rovers used a temporal-planning extension of this





Planning Graphs (Chapter 6)



- Rough idea:
  - First, solve a *relaxed problem*
    - Each "level" contains all effects of all applicable actions
    - Even though the effects may contradict each other
  - Next, do a state-space search *within the planning graph*
- Graphplan, IPP, CGP, DGP, LGP, PGP, SGP, TGP, ...

Heuristic Search (Chapter 9)

- Heuristic function like those in A*
  - Created using techniques similar to planning graphs
- Problem: A* quickly runs out of memory
  - So do a greedy search instead

- Greedy search can get trapped in local minima
  - Greedy search plus local search at local minima

- HSP [Bonet & Geffner]
- FastForward [Hoffmann]

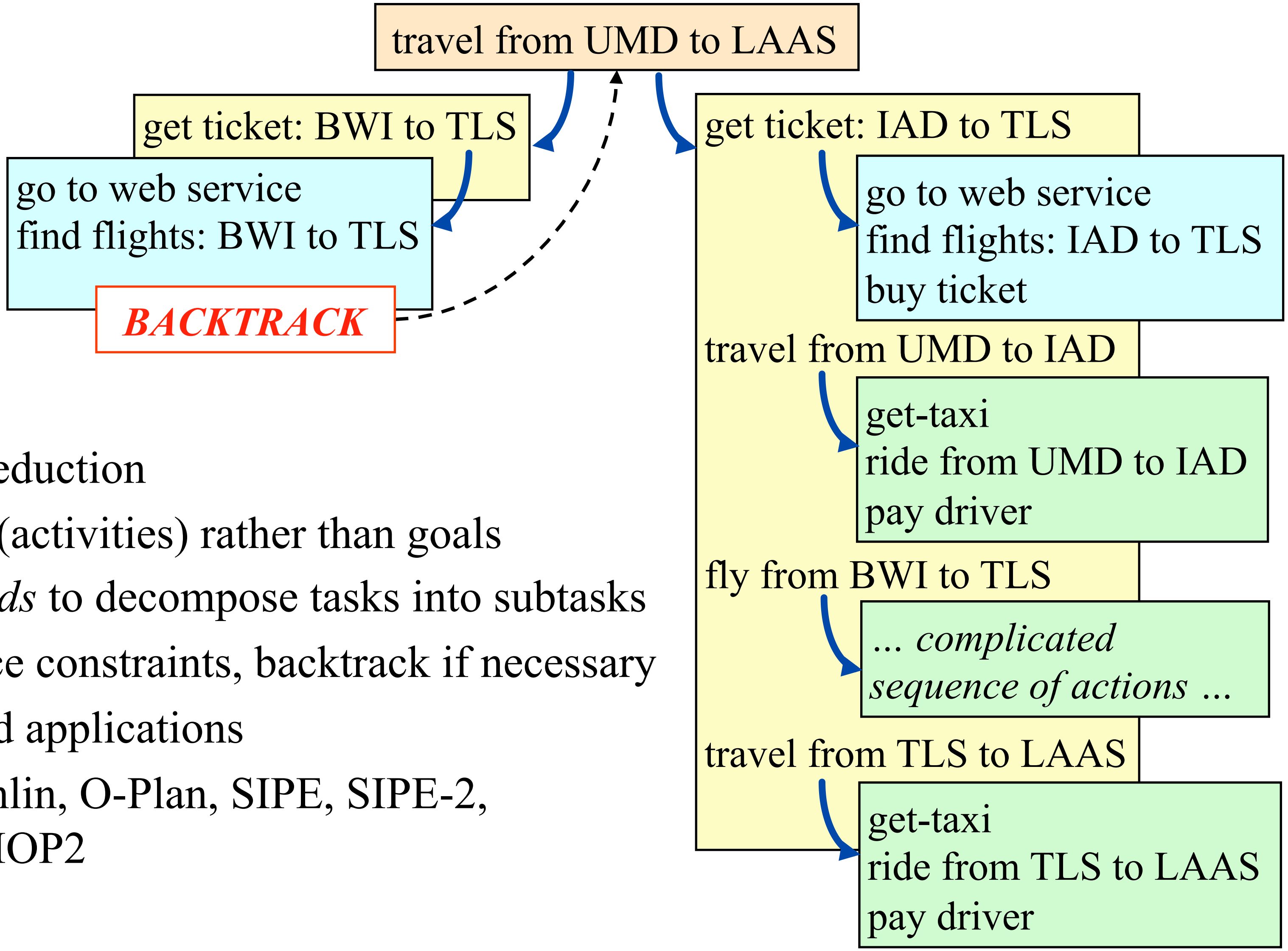
Translation to Other Kinds of Problems (Chapters 7, 8)

- Translate the planning problem or the planning graph into another kind of problem for which there are efficient solvers
  - Find a solution to that problem
  - Translate the solution back into a plan
- Satisfiability solvers, especially those that use local search
  - Satplan and Blackbox [Kautz & Selman]
- Integer programming solvers such as Cplex
  - [Vossen *et al.*]

Types of Planners: 3. Configurable

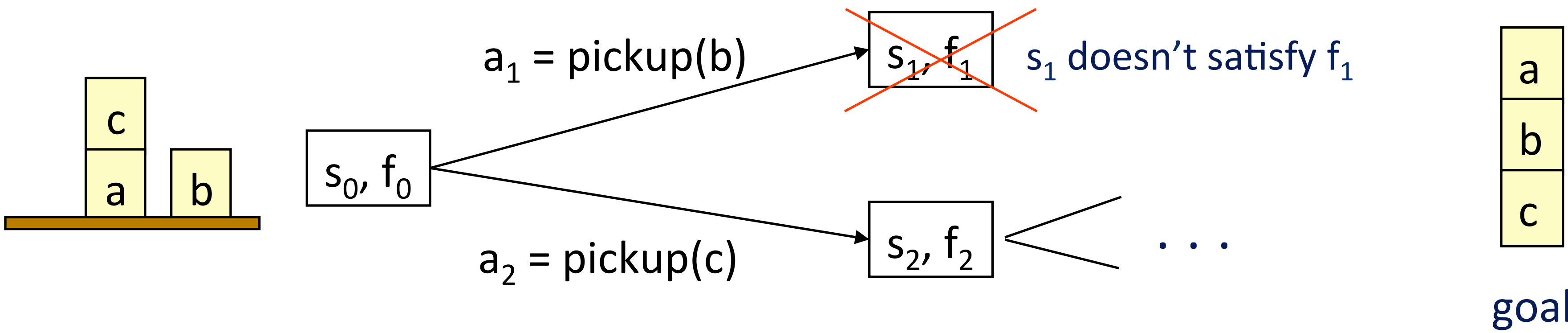
- In any fixed planning domain, a domain-independent planner usually won't work as well as a domain-specific planner made specifically for that domain
  - A domain-specific planner may be able to go directly toward a solution in situations where a domain-independent planner would explore may alternative paths
- But we don't want to write a whole new planner for every domain
- **Configurable planners**
  - Domain-independent planning engine
  - Input includes info about how to solve problems in the domain
- Generally this means one can write a planning engine with fewer restrictions than domain-independent planners
    - Hierarchical Task Network (HTN) planning
    - Planning with control formulas

HTN Planning (Chapter 11)



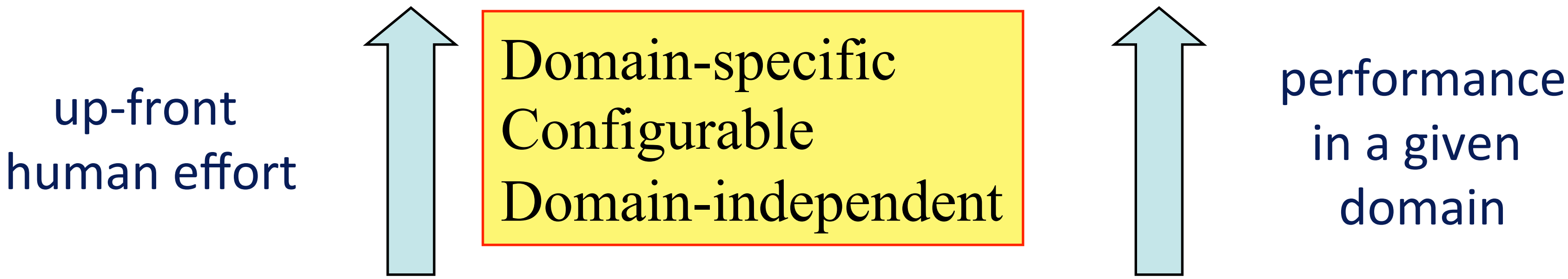
- Problem reduction
  - *Tasks* (activities) rather than goals
  - *Methods* to decompose tasks into subtasks
  - Enforce constraints, backtrack if necessary
- Real-world applications
- Noah, Nonlin, O-Plan, SIPE, SIPE-2, SHOP, SHOP2

Planning with Control Formulas (Chapter 10)



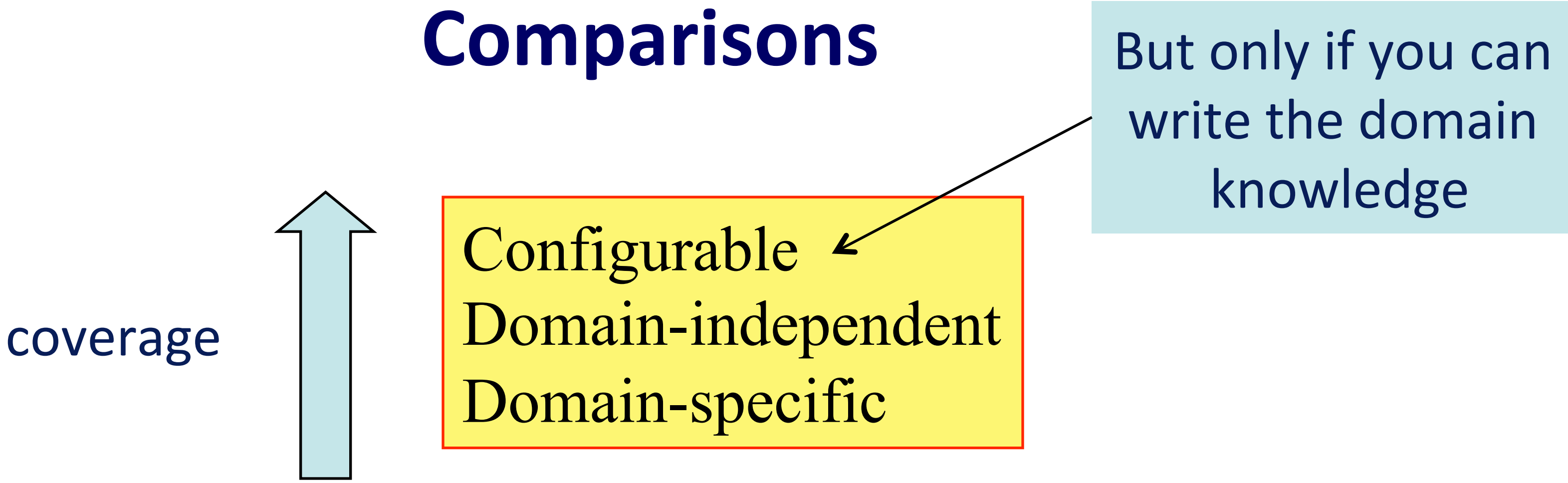
- At each state *s*, we have a *control formula* written in temporal logic
  - e.g.,
    $$ontable(x) \land \neg\exists[y{:}\mathrm{GOAL}(on(x,y))] \Rightarrow \bigcirc(\neg holding(x))$$
    "never pick up *x* unless *x* needs to go on top of something else"
- For each successor of *s*, derive a control formula using *logical progression*
- Prune any successor state in which the progressed formula is false
  - TLPlan, TALplanner, …

Comparisons



- Domain-specific planner
  - Write an entire computer program - lots of work
  - Lots of domain-specific performance improvements
- Domain-independent planner
  - Just give it the basic actions - not much effort
  - Not very efficient

Comparisons



- A domain-specific planner only works in one domain
- **In principle**, configurable and domain-independent planners should both be able to work in any domain
- **In practice**, configurable planners work in a larger variety of domains
  - Partly due to efficiency
  - Partly because of the restrictions required by domain-independent planners

Reasoning about Time during Planning

- **Temporal planning (Chapter 14)**
  - Explicit representation of time
  - Actions have duration, may overlap with each other
- **Planning and scheduling (Chapter 15)**
  - What a scheduling problem is
  - Various kinds of scheduling problems, how they relate to each other
  - Integration of planning and scheduling

Planning in Nondeterministic Environments

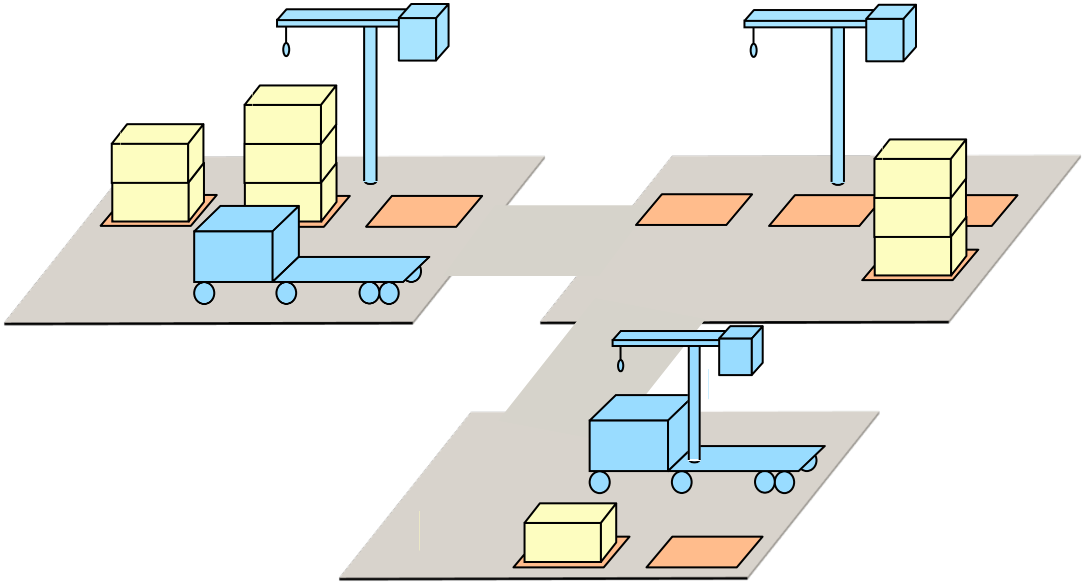
- Actions may have multiple possible outcomes
  - some actions are inherently random (e.g., flip a coin)
  - actions sometimes fail to have their desired effects
    - drop a slippery object
    - car not oriented correctly in a parking spot
- How to model the possible outcomes, and plan for them
  - **Markov Decision Processes (Chapter 16)**
    - outcomes have probabilities
  - **Planning as Model Checking (Chapter 17)**
    - multiple possible outcomes, but don't know the probabilities

Example Applications

- **Robotics (Chapter 20)**
  - Physical requirements
  - Path and motion planning
    - Configuration space
    - Probabilistic roadmaps
  - Design of a robust controller
- **Planning in the game of bridge (Chapter 23)**
  - Game-tree search in bridge
  - HTN planning to reduce the size of the game tree

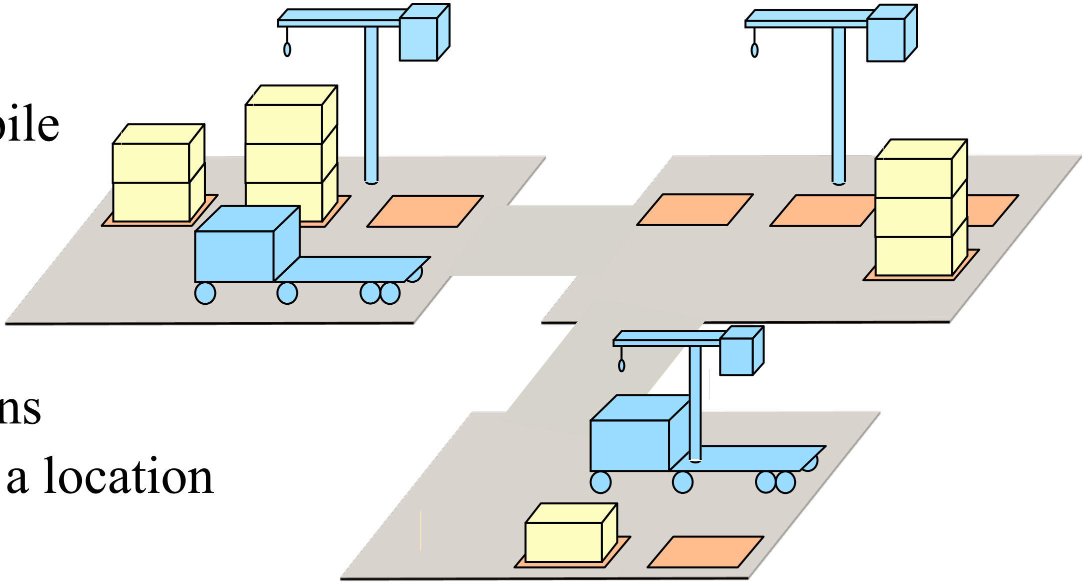
Dock Worker Robots

- Used as a source of examples throughout the book
  - A harbor with several locations
    - e.g., docks, docked ships, storage areas, parking areas
  - Containers
    - going to/from ships
  - Robot vehicles
    - can move containers
  - Cranes
    - can load and unload containers



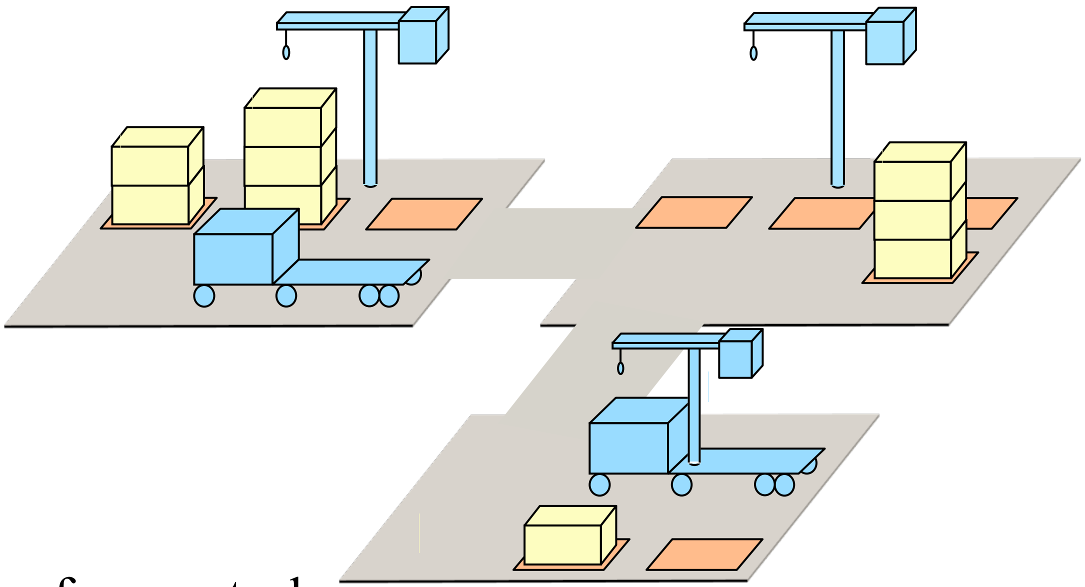
Objects

- **Locations**: l1, l2, …, or loc1, loc2, …
- **Containers**: c1, c2, …
  - can be stacked in piles, loaded onto robots, or held by cranes
- **Piles**: p1, p2, …
  - places to stack containers
  - pallet at the bottom of each pile
- **Robot vehicles**: r1, r2, …
  - carry at most one container
  - can move to adjacent locations
  - limit on how many can be at a location
- **Cranes**: k1, k2, …
  - each belongs to a single location or a single robot
  - move containers between piles and robots



Properties of the Objects

- **Rigid** properties: same in all states
  - which locations are adjacent
  - which cranes and piles are at which locations
- **Variable** properties:
  differ from one state to another
  - location of each robot
  - for each container
    - which location
    - which pile/crane/robot
    - at top of pile?



- **Actions:**
  - A crane make **take** a container from a stack,
    put it onto a stack, **load** it onto a robot, or **unload** it from a robot
  - A robot may **move** from a location to another adjacent location


Any Questions?