Automated Planning

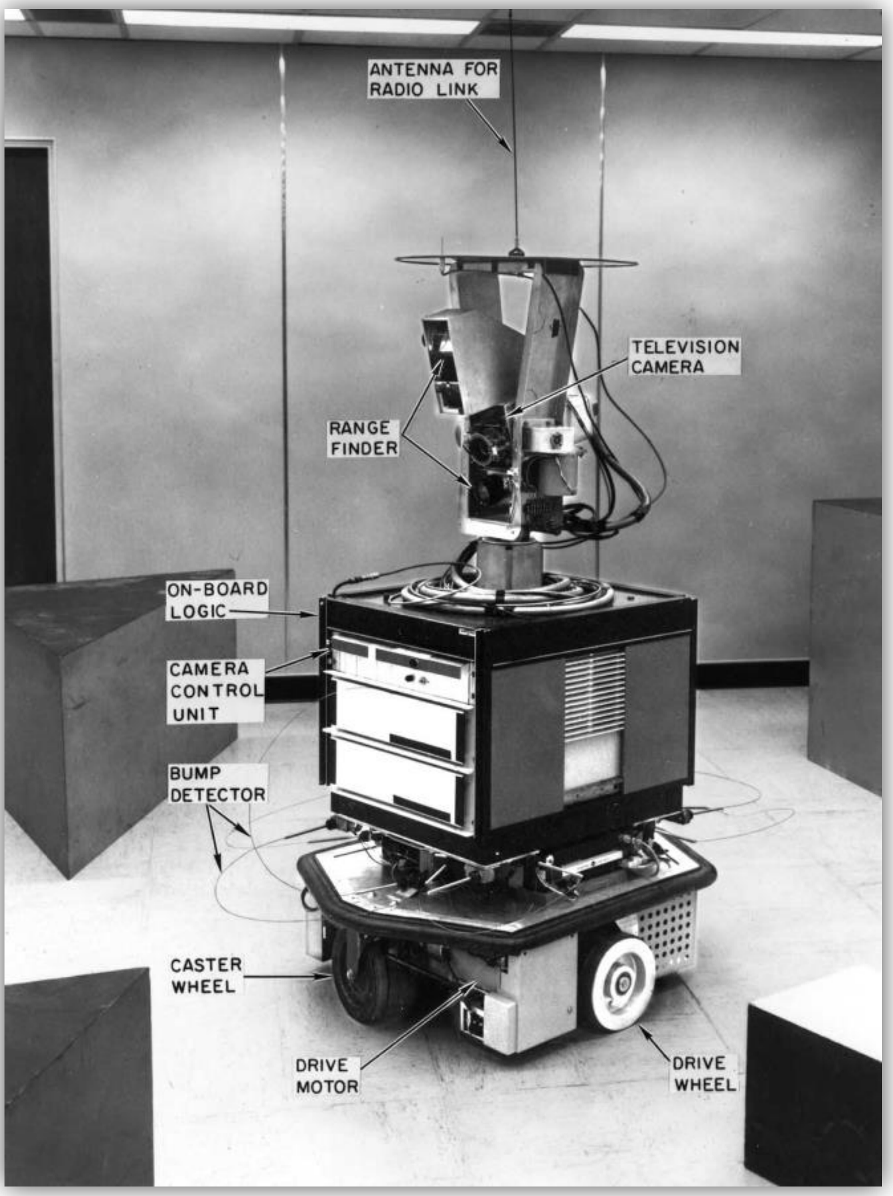
2. Classical Planning and the Planning Domain Definition Language

Jonas Kvarnström
Automated Planning Group
Department of Computer and Information Science
Linköping University

Introduction to Planning

Shakey

- Classical robot example: **Shakey** (1969)
  - Available actions:
    - Moving to another location
    - Turning light switches on and off
    - Opening and closing doors
    - Pushing movable objects around
    - …
  - Used the **STRIPS** planner
    - Stanford Research Institute Problem Solver
    - One of the first planners
    - http://www.youtube.com/watch?v=qXdn6ynwpiI

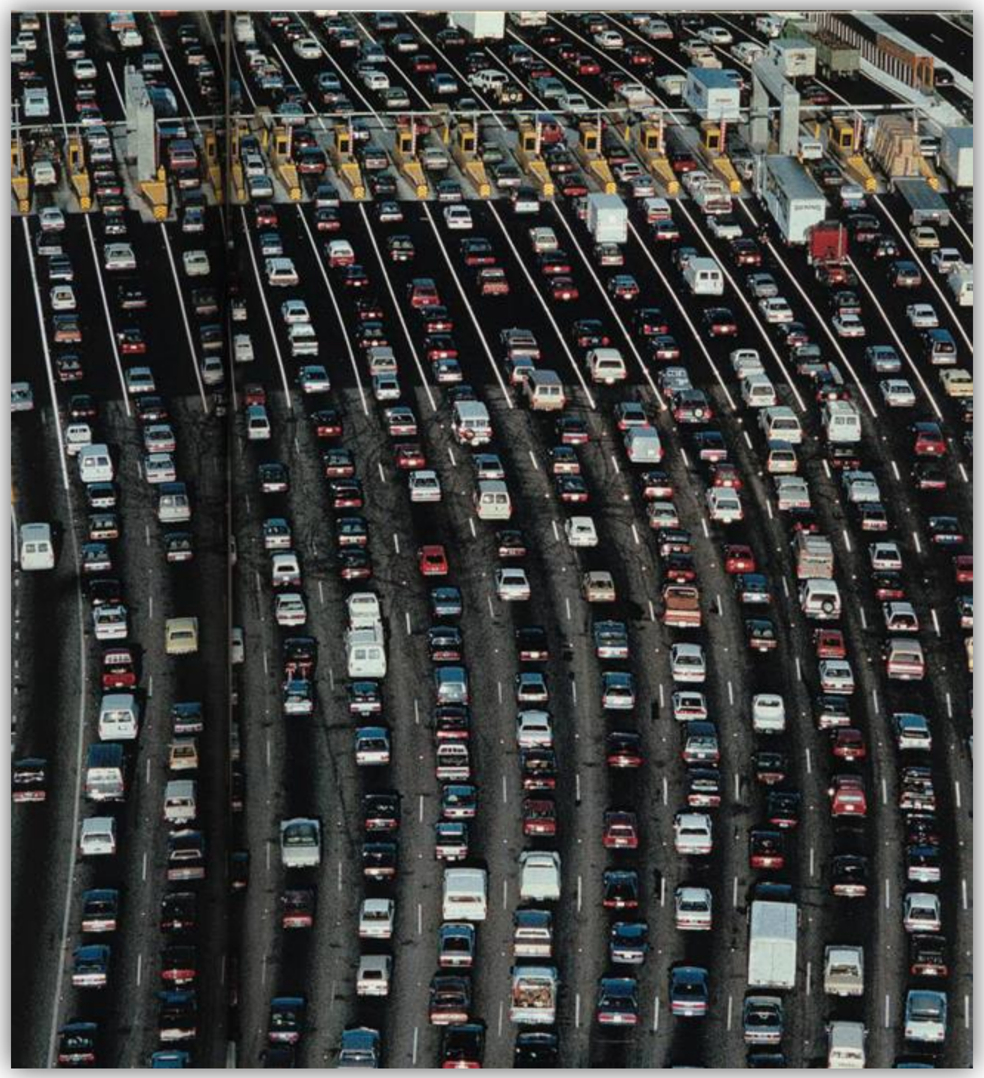


Unmanned Aerial Vehicles

- Modern robot example:
  - Autonomous
    Unmanned Aerial Vehicles (UAVs)

UAV 2: Traffic Monitoring

- Monitor traffic / find possible routes for emergency vehicles

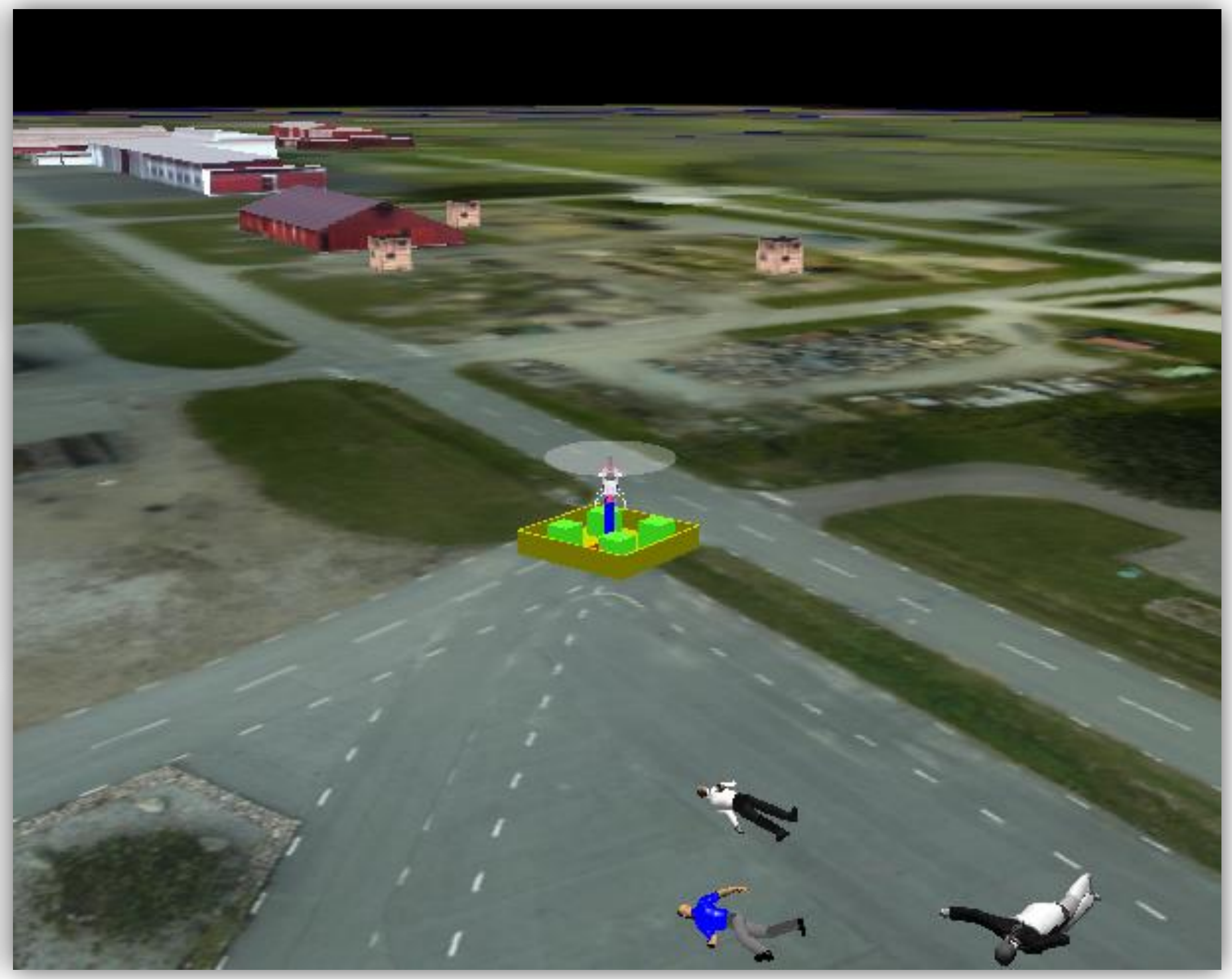


UAV 3: Finding Forest Fires

- Patrol large areas searching for forest fires, day after day after day…

UAV 4: Emergency Services Logistics

- Assist in emergency situations
  - Deliver packages of food, medicine, water

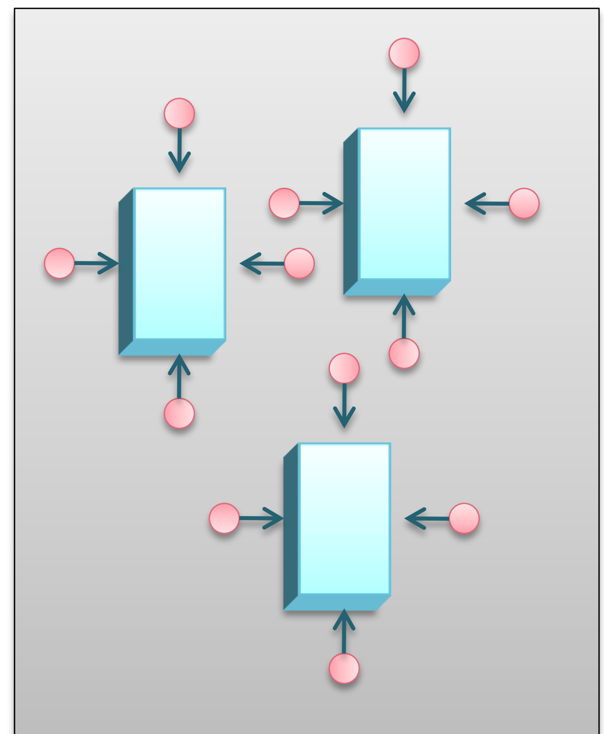


UAV 5: Photogrammetry

Photograph buildings – generate realistic 3D models

Problem Specification

- **<u>First</u>**, specify more clearly what problem you want to solve
  - We know where we want to take pictures + in which direction
  - We know how much fuel is available
  - We can *fly*, *aim* and *take pictures*
  - Aim: Determine how to take all the pictures within fuel limits!



Problem Specification

- We really have a problem *type* with many *instances*
  - We want a general solver for any instance of this type

Photogrammetry

General Problem

As specified before:
There *are* positions
(but we don't know which ones), …

Write a solver based
on this **general** information…

Problem Instance(s)

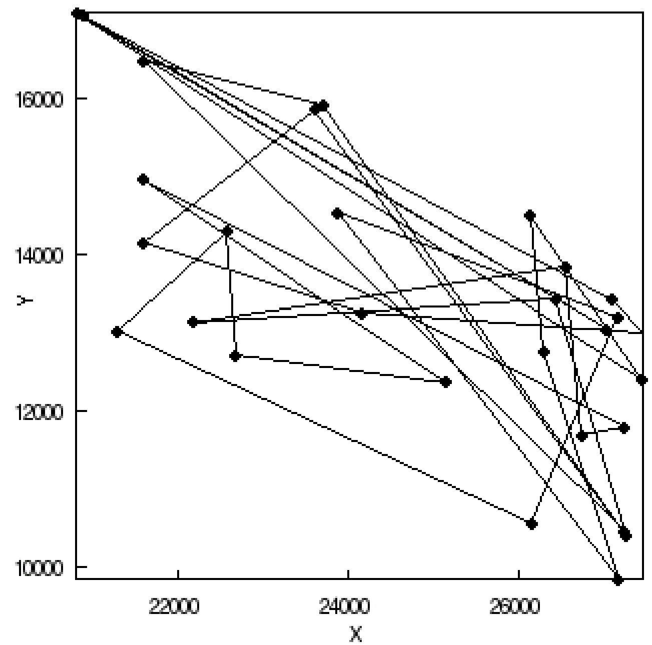
The *actual* positions and directions
the *actual* fuel level

…taking this **specific** information
as its input at runtime

Method 0: Reactive + Stupid

- Method 0: Let's be stupid
  - **while** (exists unvisited position) {
    flyto(random unvisited position)
    aim(associated direction)
    take-picture()
    }
  - No planning!
    - Very fast *algorithm*
    - Can be somewhat suboptimal…

Figure 3.1. Western Sahara: example of random initial tour



http://nfrac.org/felix/publications/tsp/

Method 1: Reactive + Greedy

- Method 1: Let's be greedy
  - **while** (exists unvisited position) {
    flyto(<u>nearest</u> unvisited position)
    aim(associated direction)
    take-picture()
    }
  - No planning!
    - Corresponds to a form of greedy search
    - Heuristically better, but not optimal
    - Worse performance for many other problems

Often, *not thinking ahead* means
you can't even solve the problem!

(Fly too far ➔ run out of fuel;
crack an egg ➔ can't uncrack it;
…)

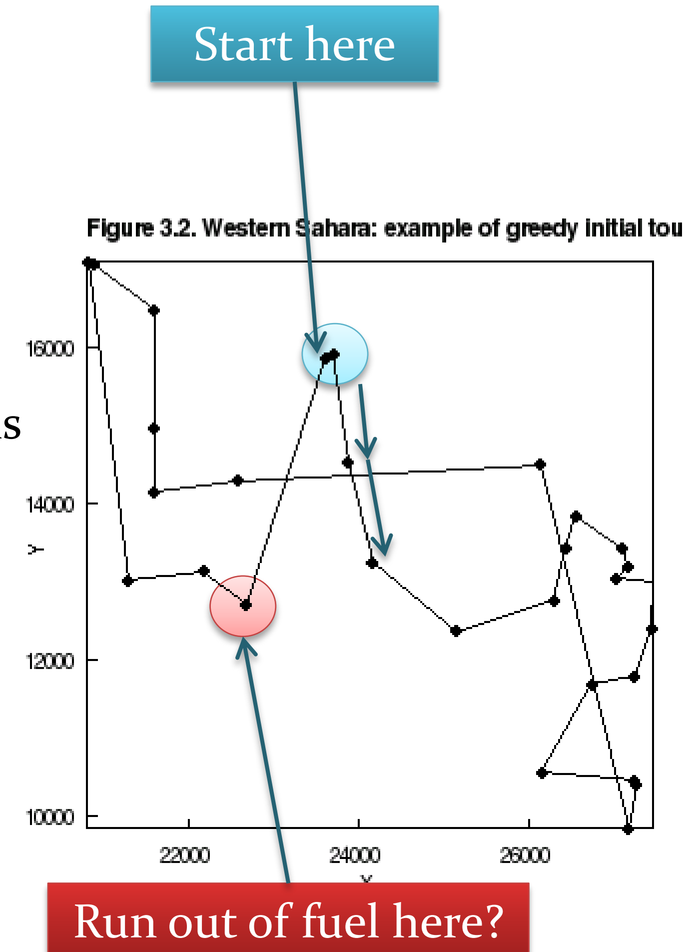


Figure 3.2. Western Sahara: example of greedy initial tour

Method 2: Think ahead

- Method 2: Let's think ahead – *first* create a complete plan
  - solve(fuel-left, current-pos, plan) {

```
solve(fuel-left, current-pos, plan) {
    if (plan visits all positions) return plan;
    foreach unvisited position pos
            in order of increasing distance
            to current-pos
    {
        f2 = fuel-left – fuel-usage(current-pos, pos);
        if (f2 > 0) {
            plan2 = solve(f2, pos, plan +
                        [flyto(pos); aim(); take-picture()]);
            if (plan2 ≠ null) return plan2;
        }
    }
    return null;
}
```

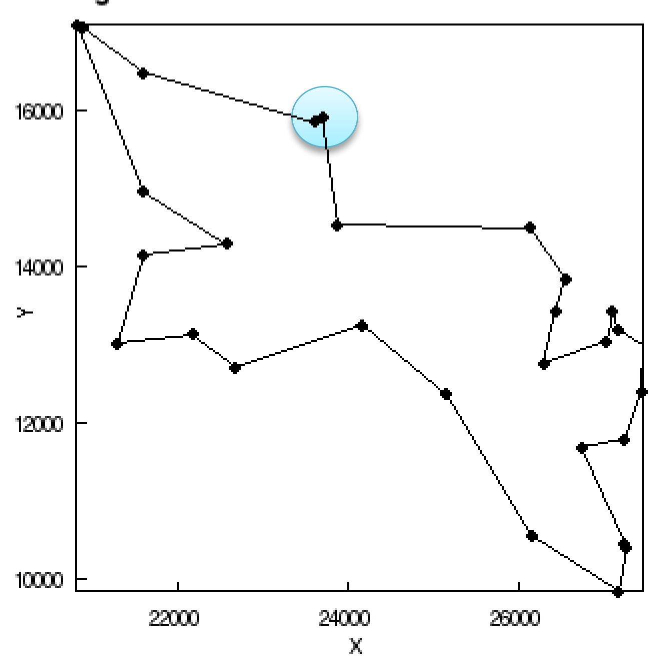
Have we already achieved the goal?

First choice: As before (*greedy heuristic*)
If not feasible: Try the *next nearest* pos

Run out of fuel "in simulation", not in reality

Backtrack if there is no feasible continuation

Figure 3.8. Western Sahara: solution tour

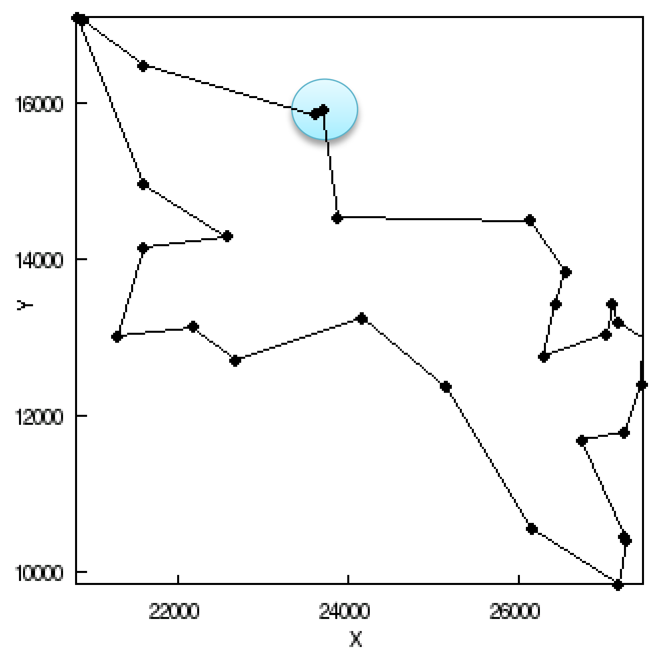


This is (a form of) planning!

Method 2: Think ahead – execution

- Method 2: Let's think ahead – *second*, execute the plan
  - **foreach** (action *a* in ordered plan) {
    execute(a)
    }

Figure 3.8. Western Sahara: solution tour



Execution is separate!

Planning:

<u>First</u> select actions, and verify they will achieve the goal
<u>Then</u> execute the solution plan – *if* a solution was found!

Comparison

Photogrammetry Without Planning

Problem spec → Generate one action (no lookahead) / Execute it / Repeat → Actions

Photogrammetry Planning

Problem spec → Photogrammetry planner → Plan → Execution system → Actions

?

Domain-Specific Planning

- So far, we have seen **domain-specific planning**
  - We identify a rather specific *type* of problem – a *planning domain*
    - *Photogrammetry planning*: given a list of locations, determine how to take pics

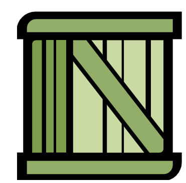
  - We analyze this problem and build a **specialized planner (solver)**
    - **A program** that can solve *all problem instances* within the domain
    - We can use all our knowledge about the domain
    - Arbitrary code – could even use a Traveling Salesman Problem (TSP) solver

The solver can be very efficient! But there are disadvantages…

Domain-Specific Planning 2

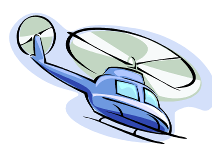
- What about more **complex problems**?
  - Efficient solutions are not as straight-forward as taking an existing TSP solver

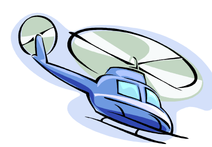
- Specialization means **less flexibility**! What if…
  - you want to **deliver** a couple of crates at the same time?
    - Need to modify the code of the planner



PG + Delivery Planner

  - you have **two UAVs** and a **UGV** (ground vehicle)?
    - Different algorithm: Multiple TSP



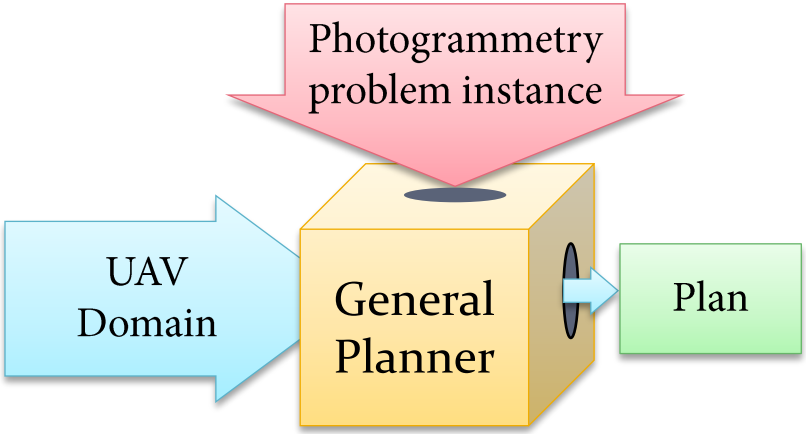


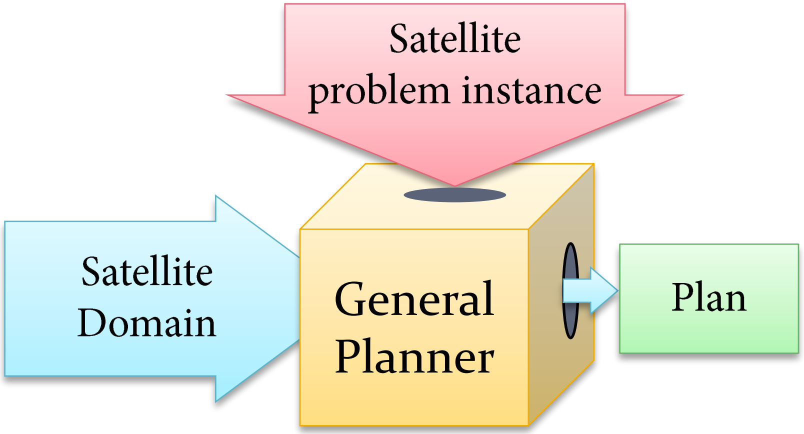
Multi-TSP planner

  - you want to survey an **area** (send video feed of the ground)?
  - you have dynamic no-fly-zones ("don't fly there at 15:00-16:00")?

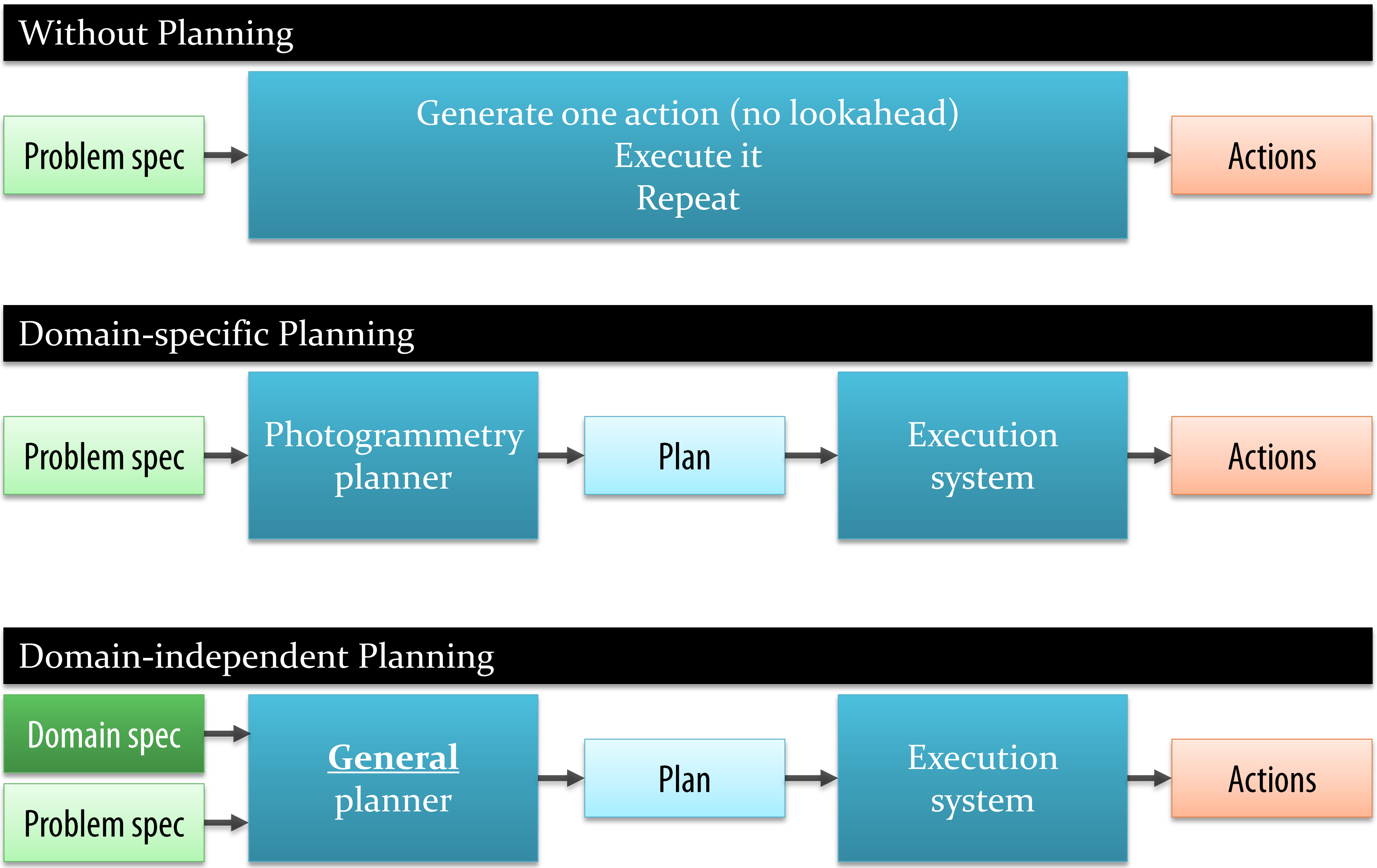
Domain-Independent Planning

- We will focus on **domain-independent** planning systems!
  - Create a **single general** planner
    - Difficult, but done *once*
    - Improvements to the planner → all domains benefit
  - Additional input: **high-level description** of a problem domain
    - Easier to specify than to write specialized algorithms
    - Easier to **change** than a hard-coded optimized implementation





Comparison 2



Planning Domains

- **Planning domain specification** for photogrammetry
  - There exist **locations**, **directions** and **helicopters**
  - The helicopter can **take off**, **land**, **fly** between locations
  - The helicopter can **aim** and **take pictures**

- **Problem instance specification** defines a problem to solve
  - In this particular problem we have:
    - **Locations** A, B, and C
    - **Directions** North, South, West and East
    - **Helicopter** H1
  - The **goal** is to have pictures
    at location A in direction North,
    …

Comparison

More effort

Higher performance

Domain-specific

Can specialize the planner for very high performance
Must write an entire planner

Domain-independent

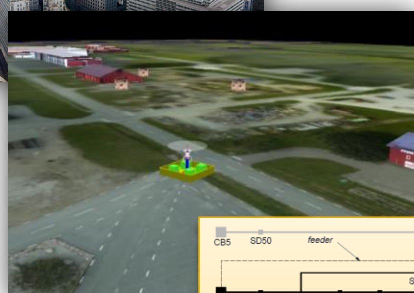
Provide high-level information
Less efficient

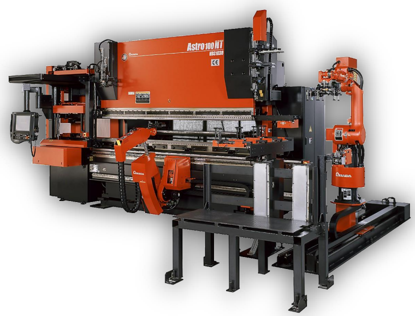
Domain-Independent: How?

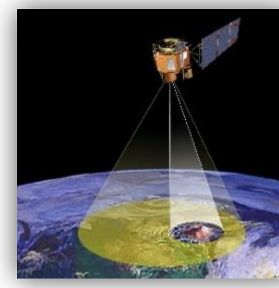
How do we create a domain-independent planner?

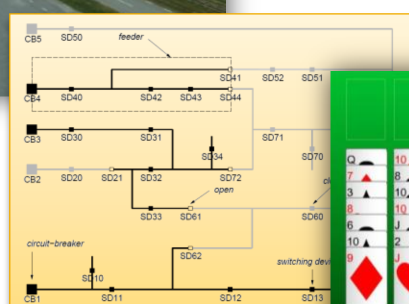
Common Properties

First, we need to find some **common concepts**
that would allow us to model a **wide variety** of domains









Common Properties

Then, we need to define…

A **formal model**
capturing
those aspects of planning domains,
instances and plans
that we consider essential

A **representation language**
allowing you to *conveniently*
describe a model

A **planning algorithm**
taking a **specification** in the representation language
and generating a **plan** satisfying the goals
according to the semantics of the formal model

Planning Domain Examples

Towers of Hanoi

Planning Domain

- There are *pegs* and *disks*
- A disk can be *on* a peg
- One disk can be *above* another

- One action:
  Move topmost disk from x to y
  - Preconditions:
    The disk must not end up *above* a smaller disk
  - Effects:
    Disk is no longer *on* x
    Disk is now *on* y

Problem Instance

- Three pegs, 7 disks
- Now: All disks on the second peg, in order of increasing size
- Goal: All disks on the *third* peg, in order of increasing size

The formal model must allow us to specify these facts!

Towers of Hanoi (2)

Planning Domain Descr.

Problem Instance Descr.

Planner →

Plan

- Move disk 1 from B to A
- Move disk 2 from B to C
- Move disk 1 from A to C
- …

A simple sequence!

Towers of Hanoi: Very restricted world!

Perfect information about all relevant facts

A single agent performing actions

A plan is simply an action sequence

…

Miconic 10 Elevators

- **<u>Tall buildings, multiple elevators</u>**
  - How to serve people as efficiently as possible?

- **<u>Schindler Miconic 10 system</u>**
  - People enter their destination *before* they board an elevator
  - A *plan* is generated, determining which elevator goes to which floor, and in which order
  - Saves time!

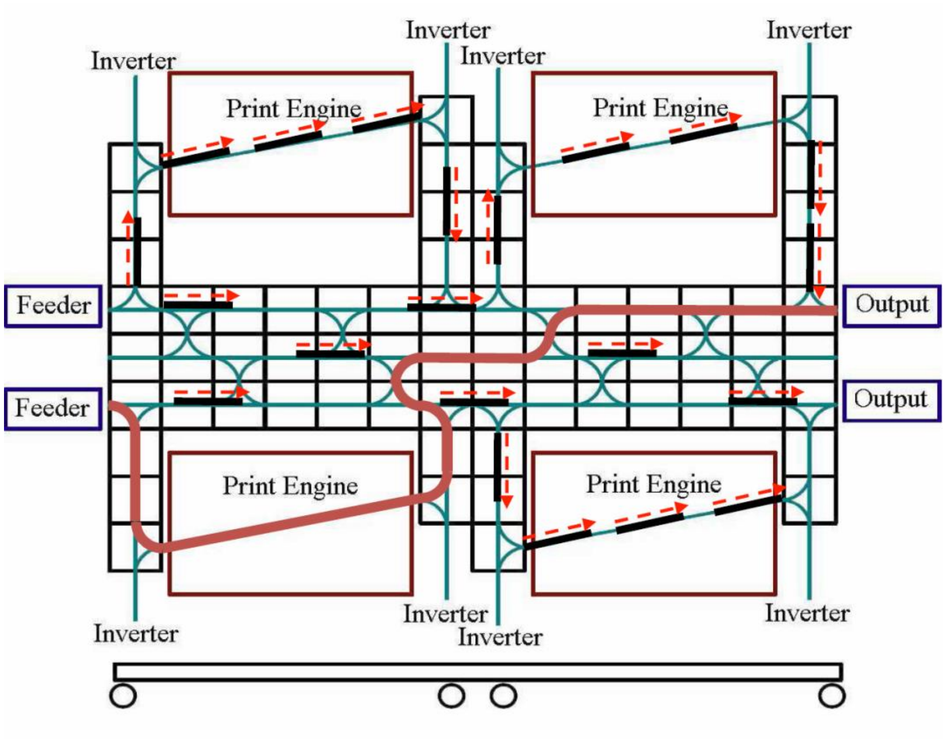
Comparison

- Single agent,
  one action at a time
- All actions take approximately the same amount of time

- Several agents (the elevators),
  concurrent actions
- Timing differs
  (and is essential for quality):
  Going from floor 1 to 3,
  or from floor 1 to 99?

Xerox Printers

- Xerox: Reconfigurable modular printers
  - Prototype: 170 individually controlled modules, 4 print engines
  - Goal: Finish each print job as quickly as possible



Comparison

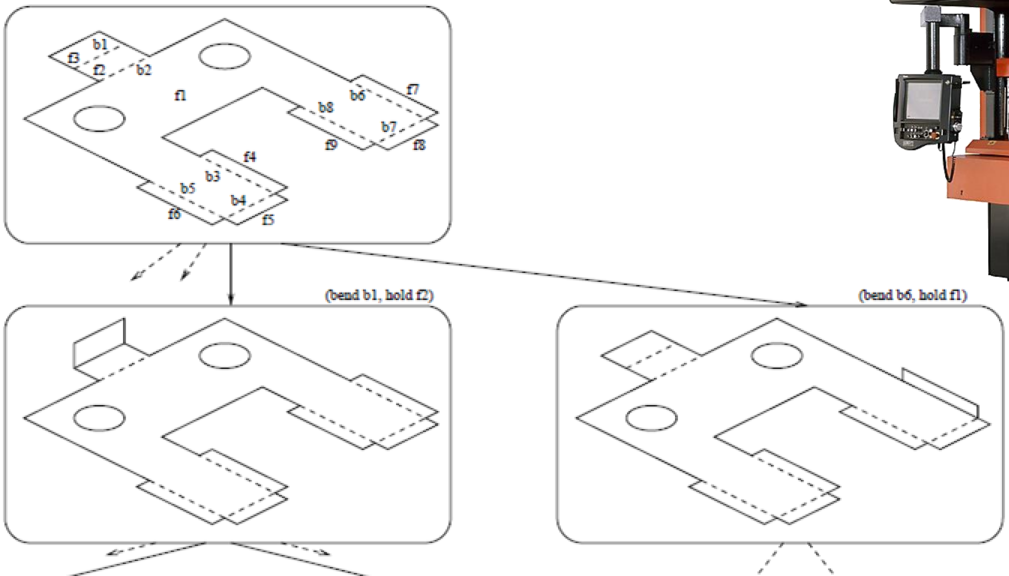
- Concurrency:
  - Useful for performance
  - If we miss an opportunity:
    Lower quality

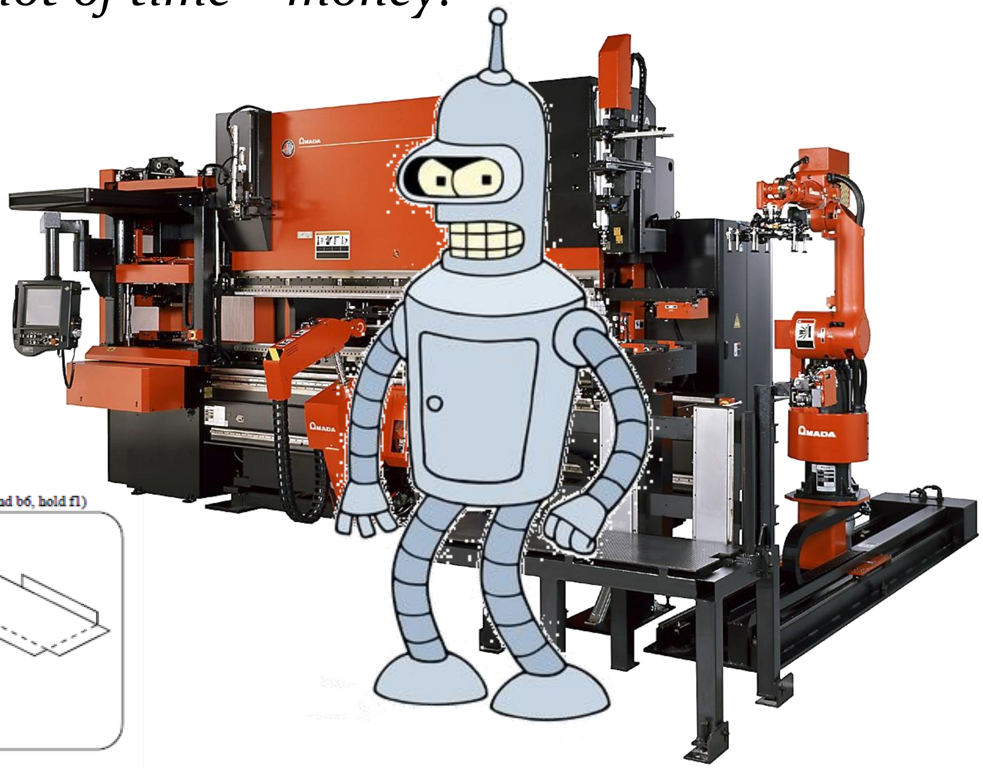
- Concurrency:
  - Necessary for correctness
  - If we miss:
    Paper jam, ...

Bending Sheet Metal

- Bending sheet metal
  - Goal: Bend a flat sheet to a specific shape
  - Constraints: The piece must not collide with anything when moved!

  - ***Optimized operation*** *saves a lot of time = money!*





Comparison

- Might use metric values
  - Distances, timing

- Need 3D geometry
  - Current state
  - Preconditions: Will the piece fit in a certain configuration?
  - Effects: Reason about bending, …

DARPA Grand Challenge 2005

- Competition: **autonomous cars** drive 212 km off-road

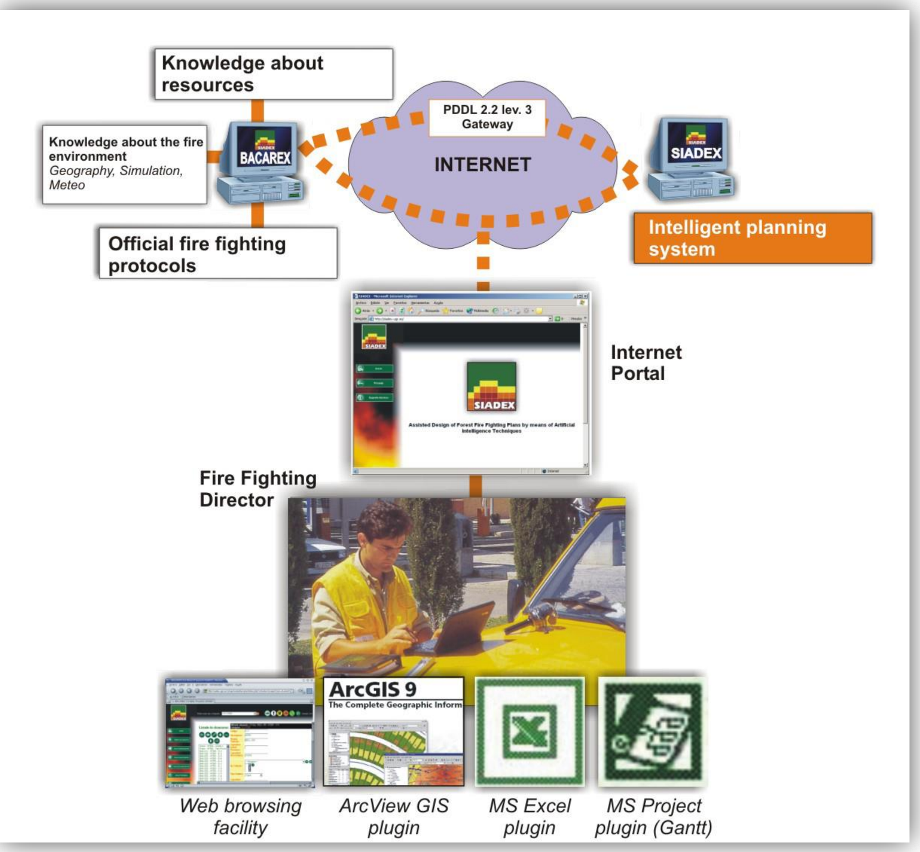
- Requires **path planning**
  - Deciding how to get from one point to another, given:
    - Speed limits
    - Constraints on how you can move (turn radius, …)
    - A map – that may not always be correct
    - http://www.youtube.com/watch?v=M2AcMnfzpNg 2:00, 4:00

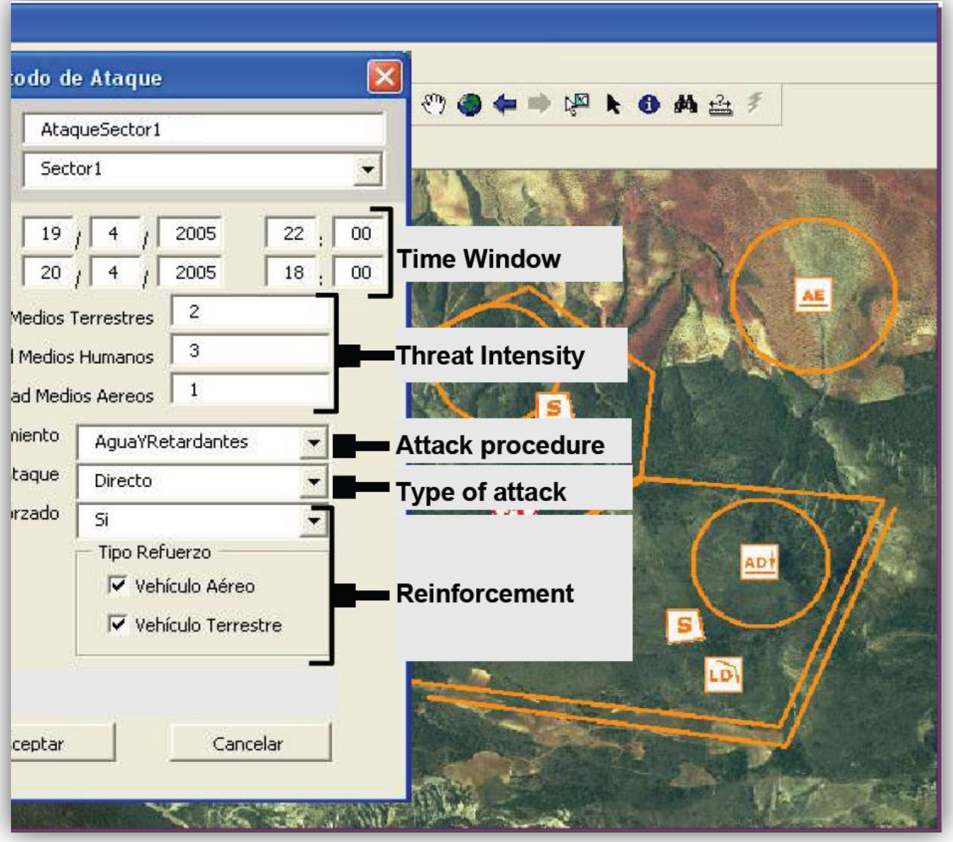
DARPA Urban Challenge 2007

- Competition: 96 km in an urban area (air force base)
  - Must follow all traffic regulations, drive around obstacles, merge with other traffic, …

SIADEX

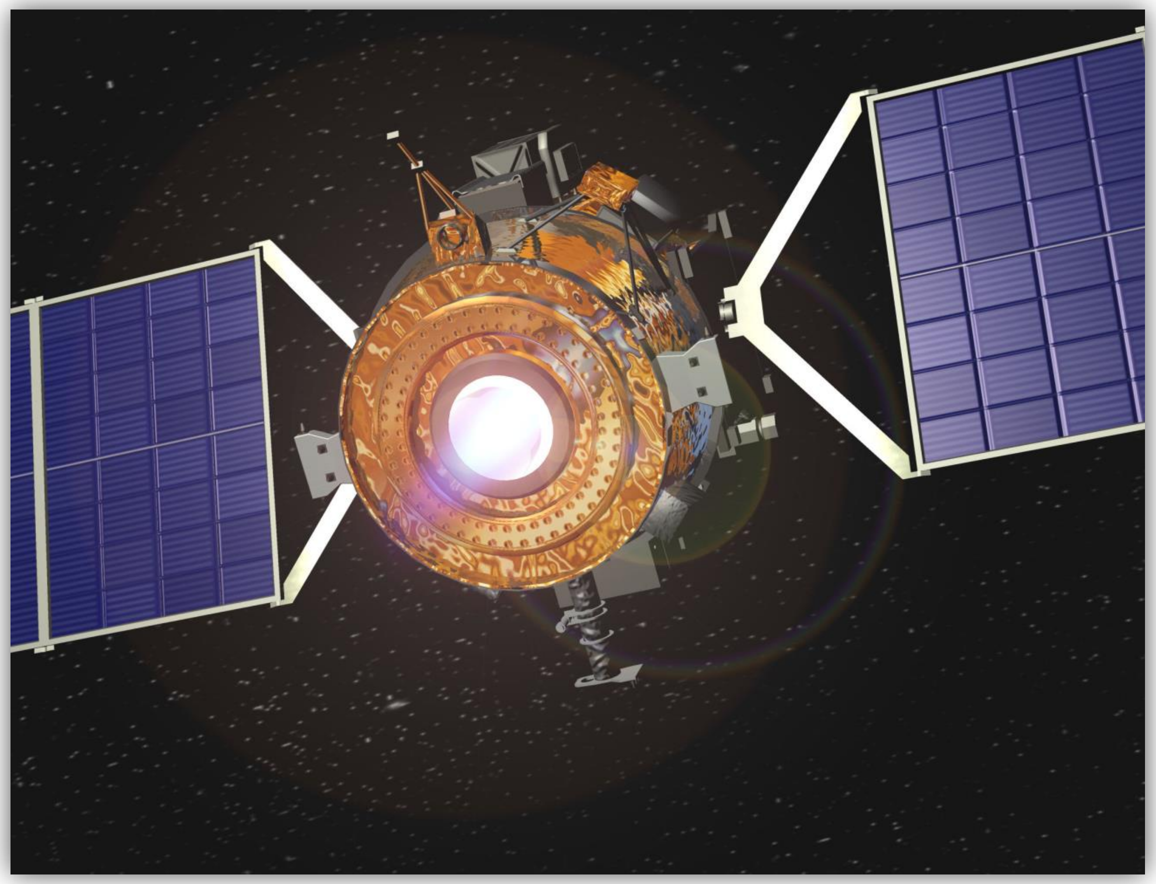
- SIADEX
  - Decision support system for designing **forest fire fighting** plans
  - *Needs to consider allocation of* ***limited resources***
  - *Plans must be developed in* ***cooperation*** *with humans –* ***people may die****!*





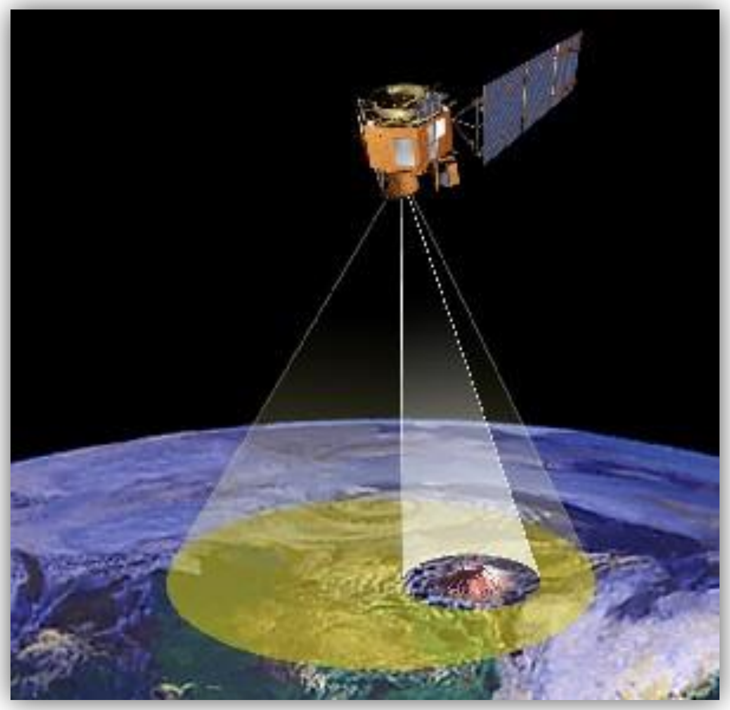
NASA Remote Agent

- ”Remote Agent” on Deep Space 1 spacecraft
  - Experimental online operation for 2 days
  - Correctly handled simulated failures
  - Rapid response to failures may be crucial to survival!

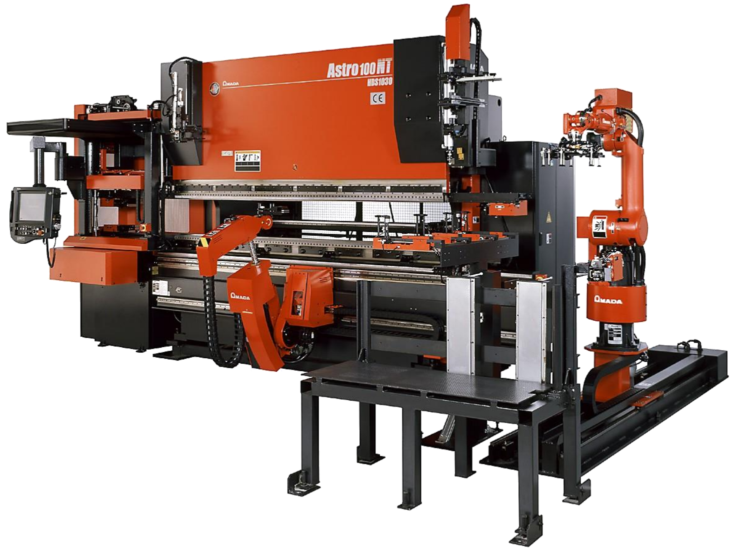


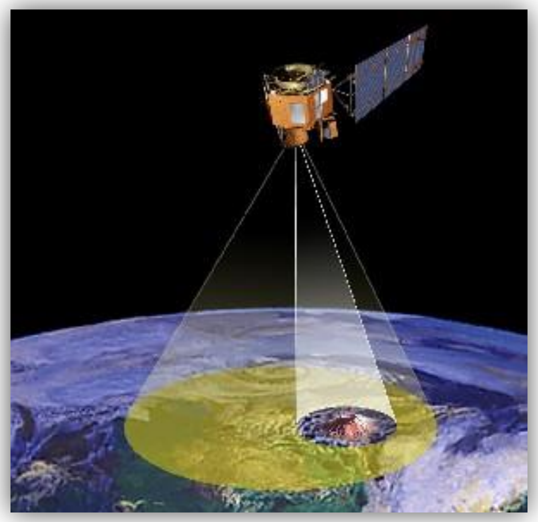
NASA Earth Observing-1 Mission

- Earth Observing-1 Mission
  - Satellite in low earth orbit
    - Can only communicate 8 x 10 minutes/day
    - Operates for long periods without supervision
  - CASPER software:
    Continuous Activity Scheduling, Planning, Execution and Replanning
  - **Dramatically increases science returns**
    - Interesting events are analyzed (volcanic eruptions, ...)
    - Targets to view are planned depending on previous observations
    - Plans downlink activities: Limited bandwidth
  - **http://ase.jpl.nasa.gov/**



Comparison





- All goals are given in advance
- Achieve all goals

- New goals may arrive, must reconsider the plan
- Can’t achieve all goals – must **<u>prioritize</u>**

Additional Issues

Various issues

Incomplete information:
We know about some obstacles, might discover others during execution
Must take new facts into account!

Agents involved in **other activities / multiple plans**:
May already be busy at some times

Self-interested agents:
Must negotiate about actions to be performed

…

Why should Computers Plan?

- Now we see why we want **<u>computers</u>** to create plans:

  - Manual planning can be **<u>boring</u>** and **<u>inefficient</u>**
    - Who wants to spend all day guiding elevators?

  - Automated planning may **<u>create higher quality plans</u>**
    - Software can systematically optimize, can investigate millions of alternatives

  - Automated planning can be applied **<u>where the agent is</u>**
    - Satellites cannot always communicate with ground operators
    - Spacecraft or robots on other planets may be hours away by radio

Common Properties

Can we now find…

Very difficult to specify a well-defined semantics
for the *combination* of all of these requirements

A **single** formal model?

A **single**
representation language?

A **single** planning algorithm
capable of generating a plan
in any of these domains?

Extremely difficult to find an algorithm
that works well in all of these situations

Common Properties (2)

- A planner should also:
  - Generate plans as **<u>quickly</u>** as possible
  - Generate plans of the **<u>highest quality</u>** possible
    - Fewer actions, lower cost, faster to execute, …
  - **<u>Support the user</u>** as much as possible
    - Provide useful high-level structures such as actions that a user can easily specify

Conflicting desires – we need trade-offs!

There are many *different* tradeoffs
that have proven useful…

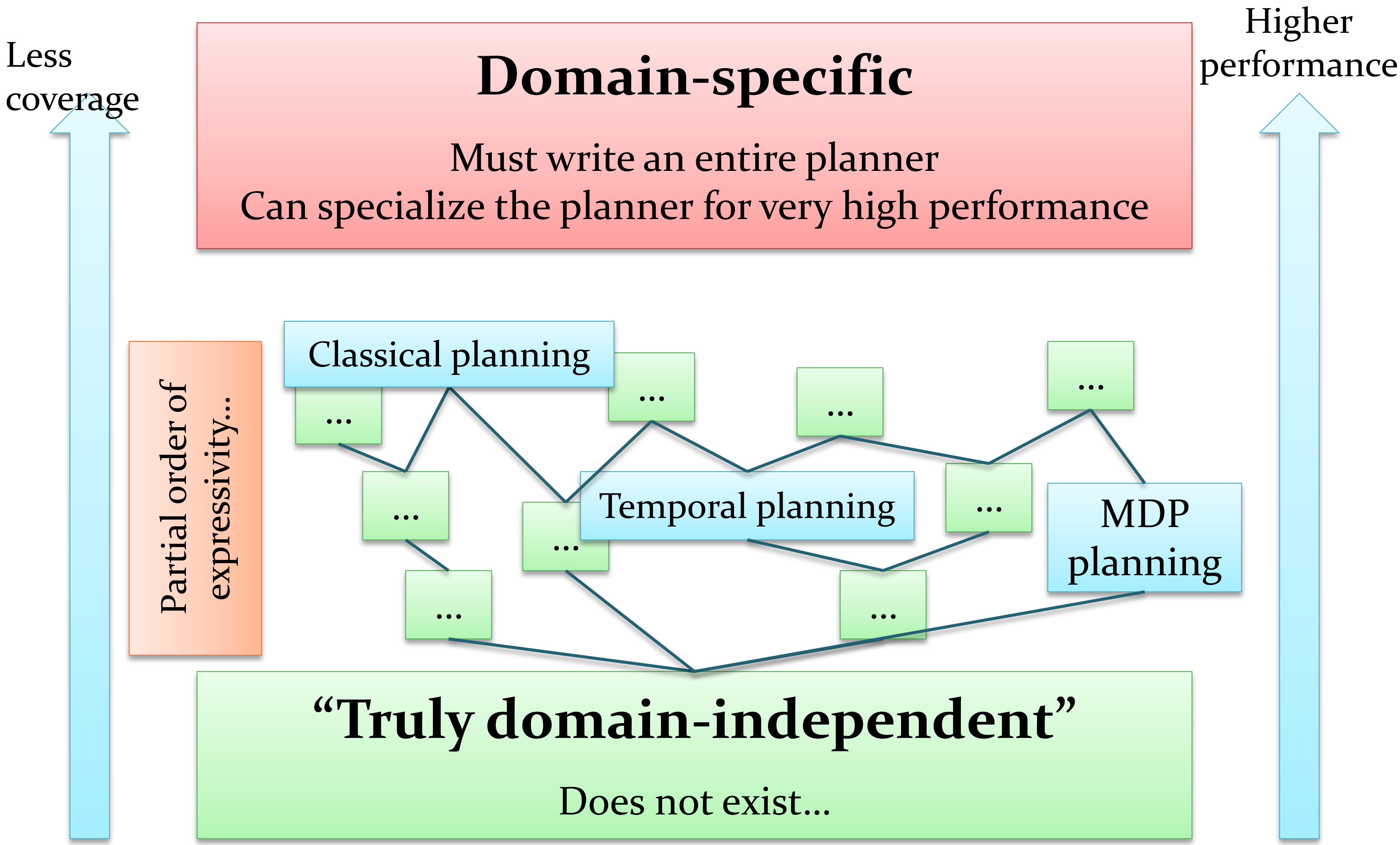
Levels of Expressivity

No planner is truly "domain-independent"
in the sense that it accepts **every** planning problem
you can think of

No planner is more expressive than all other planners

Decide what "kind" of domains your planner should be able to accept
Write a planner for this **expressivity**
Use the restrictions you have to improve performance

Levels (2)


Less coverage
Higher performance
Domain-specific
Must write an entire planner
Can specialize the planner for very high performance
Partial order of expressivity…
Classical planning
…
…
…
…
…
…
…
Temporal planning
…
…
MDP planning
…
"Truly domain-independent"
Does not exist…

Classical Planning

Classical Planning

- Many early planners were similar in terms of…

The expressivity of the **formal model**

The expressivity of the **representation language**

The associated **assumptions** about the world

- We often call this **classical planning**
  - Quite restricted, but we have to start somewhere…
    - Forms the basis of most non-classical planners as well
  - Some disagreement on **exactly** how this should be defined
    - The definition in the book (and here)
      shows the *essence* of what classical planning means

There are many non-classical planners as well!

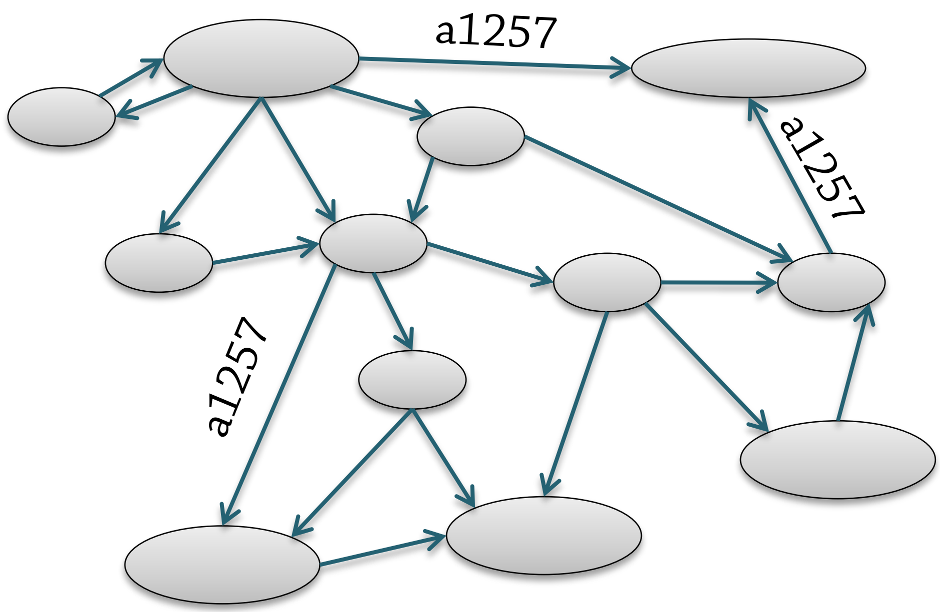
Classical Planning (2)

- In classical planning, the world **can be described** as having:

A0

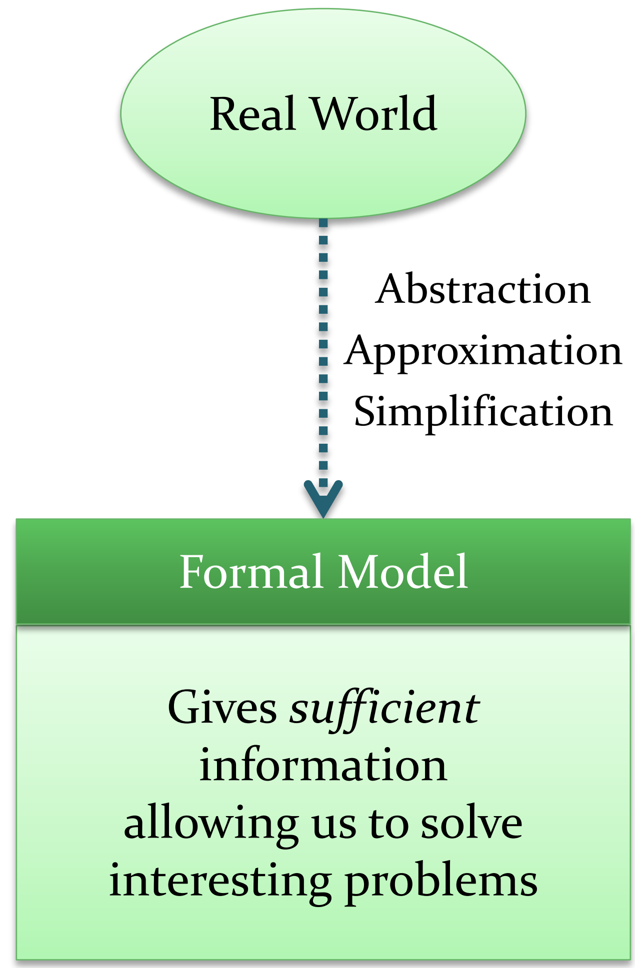
  - A finite set of **states**
  - A finite set of **actions** that take you between states
    - The **outcome** can depend on the state in which the action was started

Assumption number in the course book



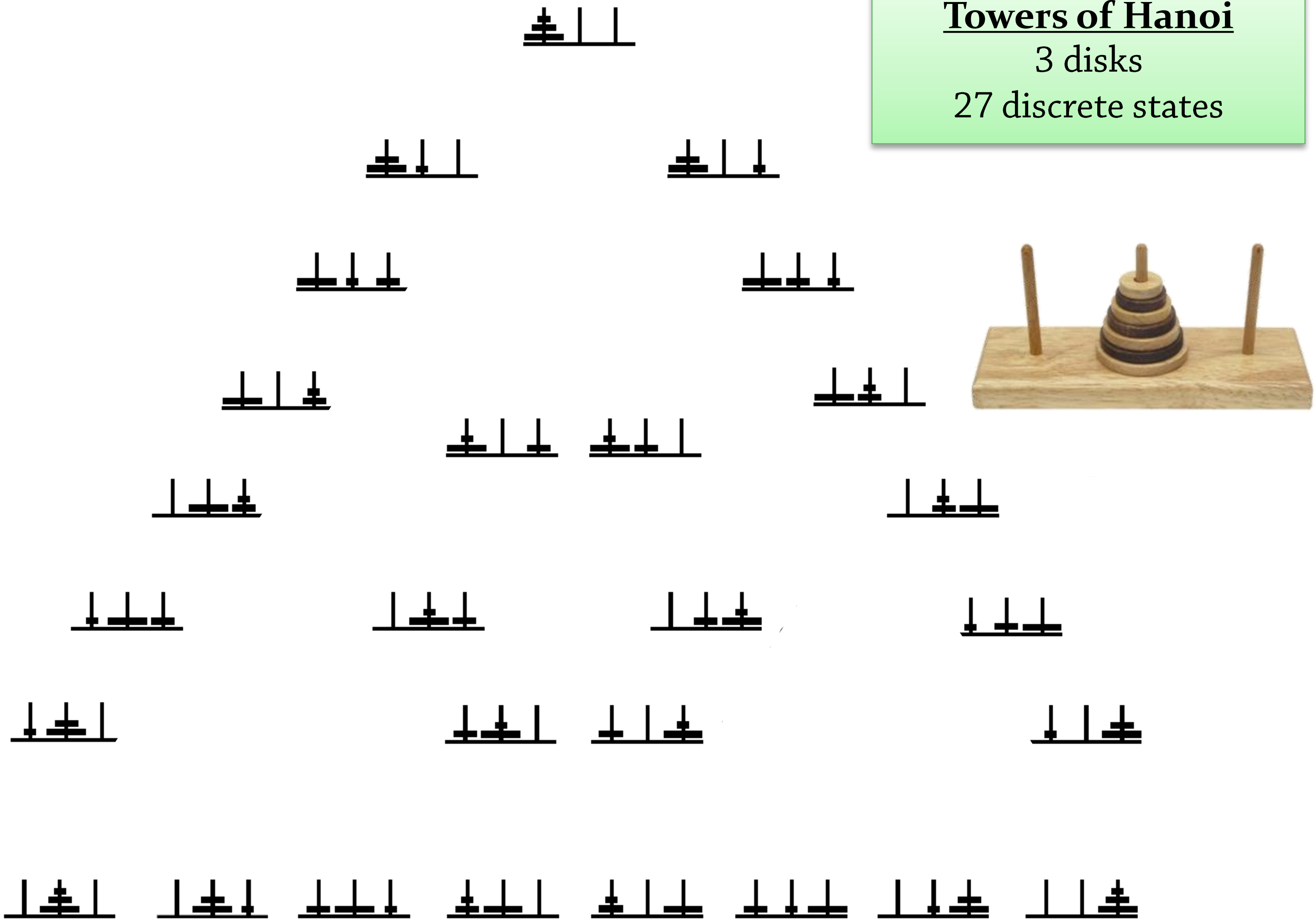
Classical Planning (3)

- Note: ”**<u>can be described as</u>**”
  - Towers of Hanoi: Disks can be placed *continuously* in 3D space
    - Uncountably infinite number of states, actions
  - But for the purpose of **<u>planning</u>**:
    - Finite number of *interesting* states and actions



Classical Planning (4)

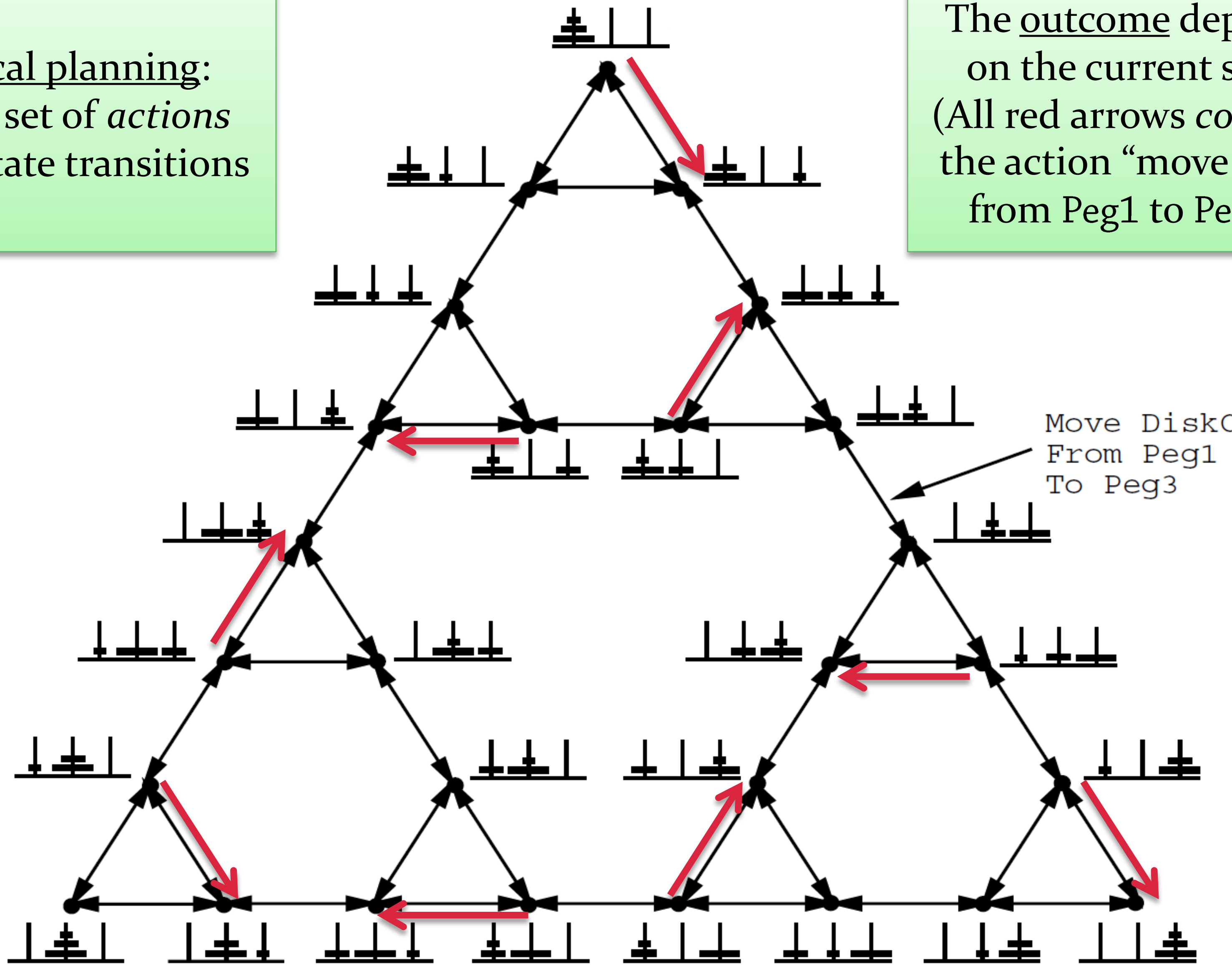
Towers of Hanoi
3 disks
27 discrete states



Classical Planning (5)

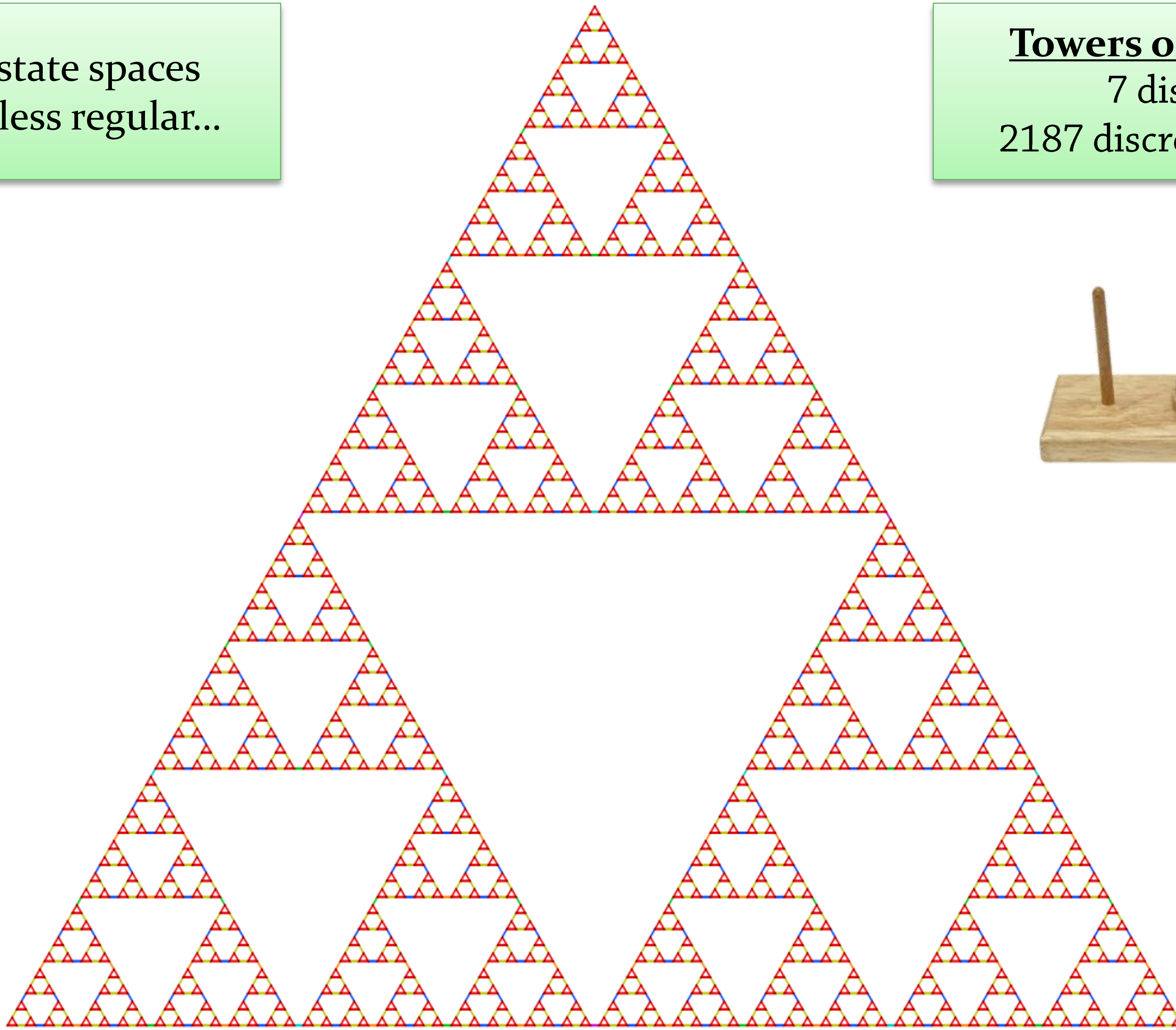
Classical planning:
A finite set of *actions*
induce state transitions

The outcome depends
on the current state
(All red arrows *could* be
the action "move diskA
from Peg1 to Peg3")



Classical Planning (6)

Most state spaces are far less regular...



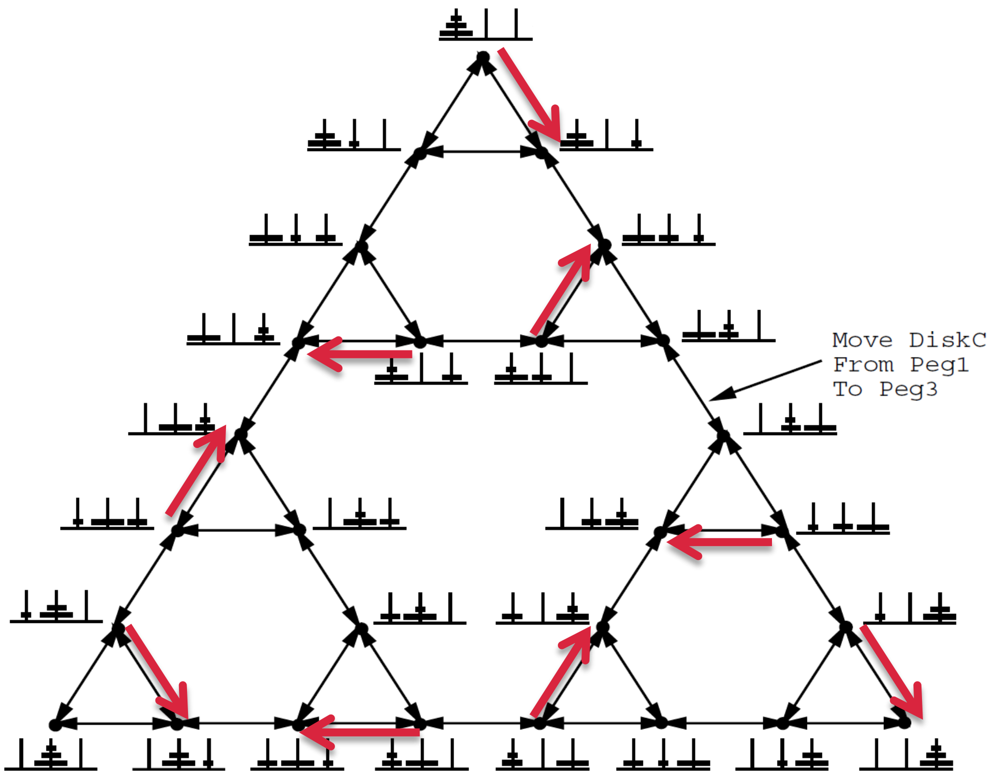
Towers of Hanoi
7 disks
2187 discrete states

Classical Planning (7)

- In classical planning, we assume:
  - A6 **Temporal aspects** of actions can be **ignored**
    - We don't model or care about time requirements
    - For the purpose of planning,
      the transition between two states has no duration

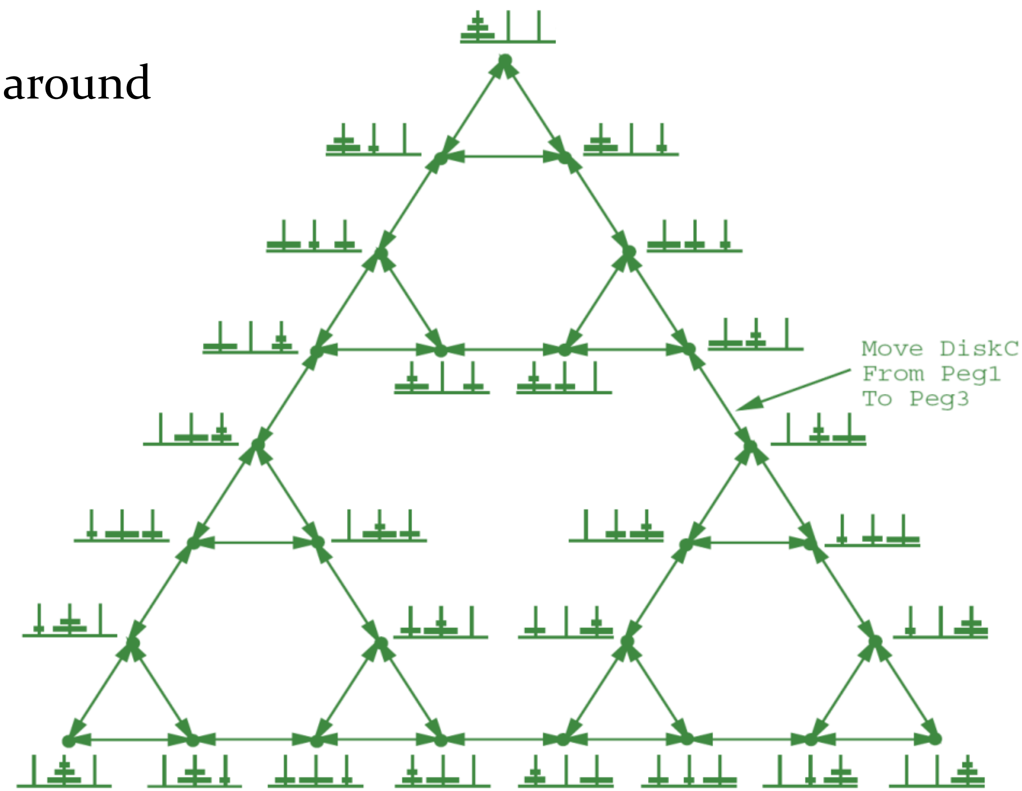
Towers of Hanoi
3 disks
27 states

The correct solution does not depend on the time to move a disk, the weight of the disk, …



Classical Planning (8)

- Additional assumptions:
  - A2 Each action is **deterministic**
    - If we know which state we are in and which action we execute, we know which state we end up in
  - A3 The world *only* changes state when we execute one of these actions
    - No spontaneous change
    - No other agents running around and making changes



Restricted State Transition System

- Formally: a **restricted state transition system** $\Sigma = (S, A, \gamma)$
  - $S = \{ s_0, s_1, \ldots \}$: Finite set of **world states**
  - $A = \{ a_0, a_1, \ldots \}$: Finite set of **actions**
  - $\gamma: S \times A \rightarrow 2^S$: **State transition function**, where $|\gamma(s,a)| \leq 1$
    - If $\gamma(s,a) = \{s'\}$, then whenever you are in state *s*, you can execute action *a* and you end up in state *s'*
    - If $\gamma(s,a) = \emptyset$ (the empty set), then *a* cannot be executed in *s*

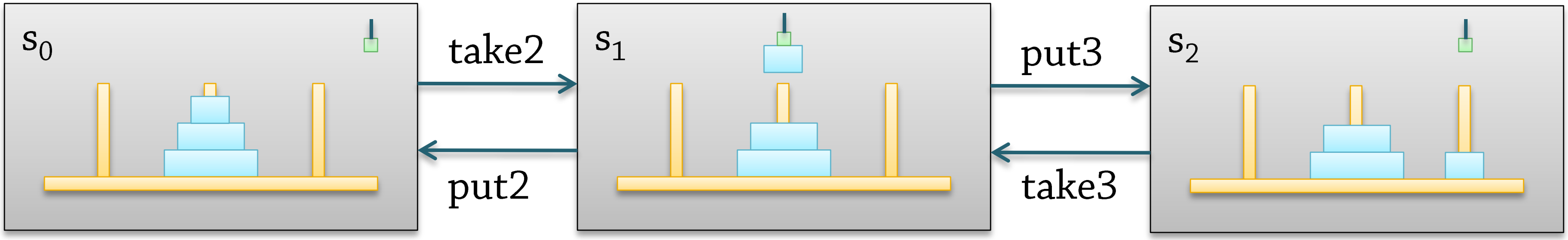
$S = \{ s_0, s_1, \ldots \}$

$A = \{ take1, put1, \ldots \}$

$\gamma: S \times A \rightarrow 2^S$

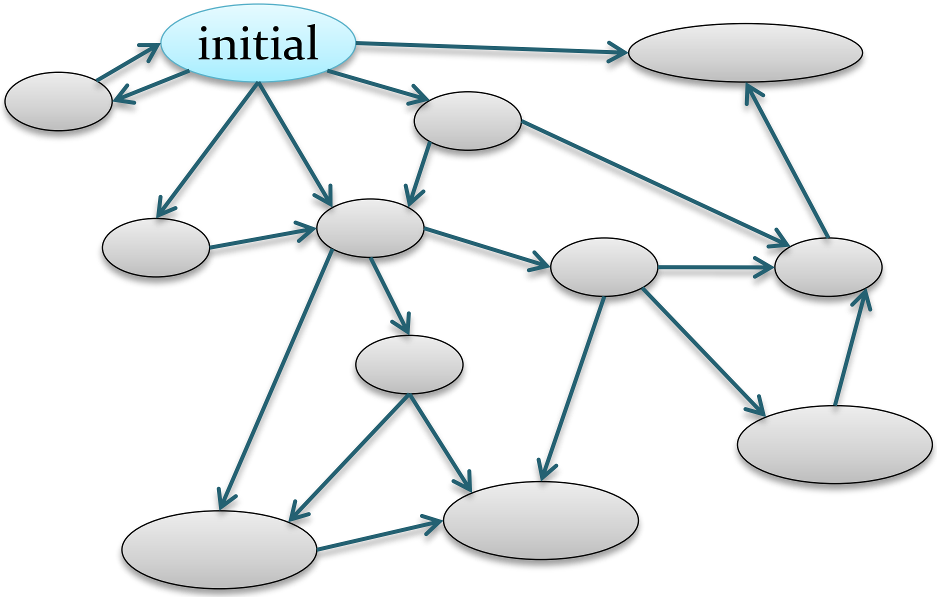
$\gamma(s_0, take2) = \{ s_1 \}$

$\gamma(s_1, take2) = \emptyset$



Classical Planning (9)

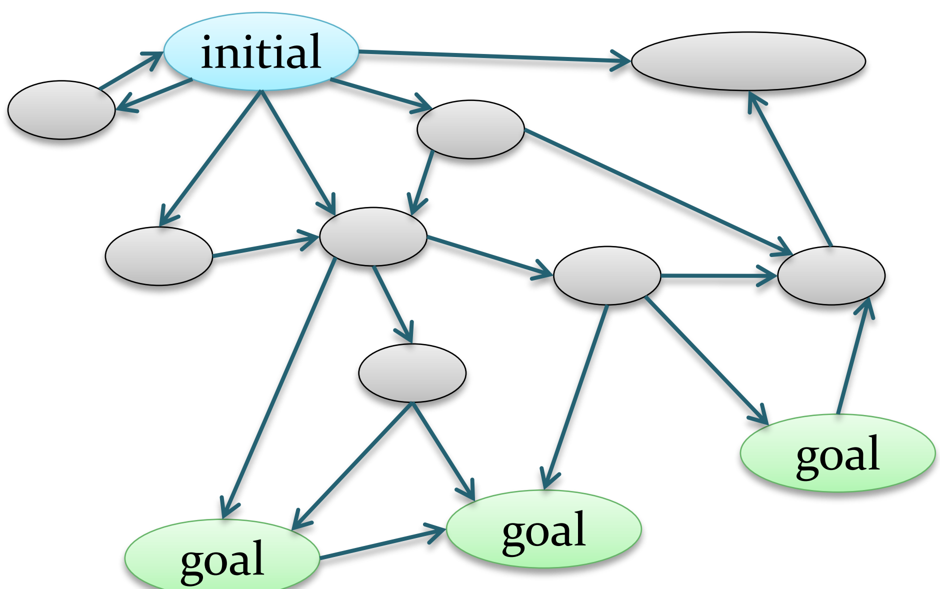
- Assumptions:
  - A1 We always know the **current state** of the world
  - A7 The world **does not change** while we're generating plans
    - So if we check which state we're in now,
      then we generate a plan,
      we will still be in that state when we start executing the plan
    - We know the initial state!



Classical Planning (10)

- Assumptions:
  - A4 Our objective is to transform the world so that we end up in any of a set of **goal states**
    - How we reach one of these states is irrelevant

In *non-classical* planning, our objective could include:
Achieving a goal in a certain amount of time,
Visiting interesting states along the way / *not* visiting dangerous states,
…



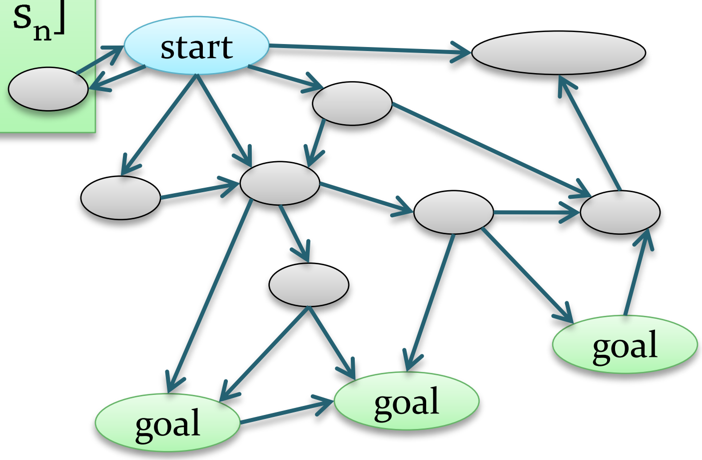
Classical Planning (11)

- Assumptions:
  - A5 A plan is simply a **sequence** of actions
    - Actions cannot be executed in parallel
    - Deterministic, no exogenous actions → no need for if-then conditions

Classical Planning Problem

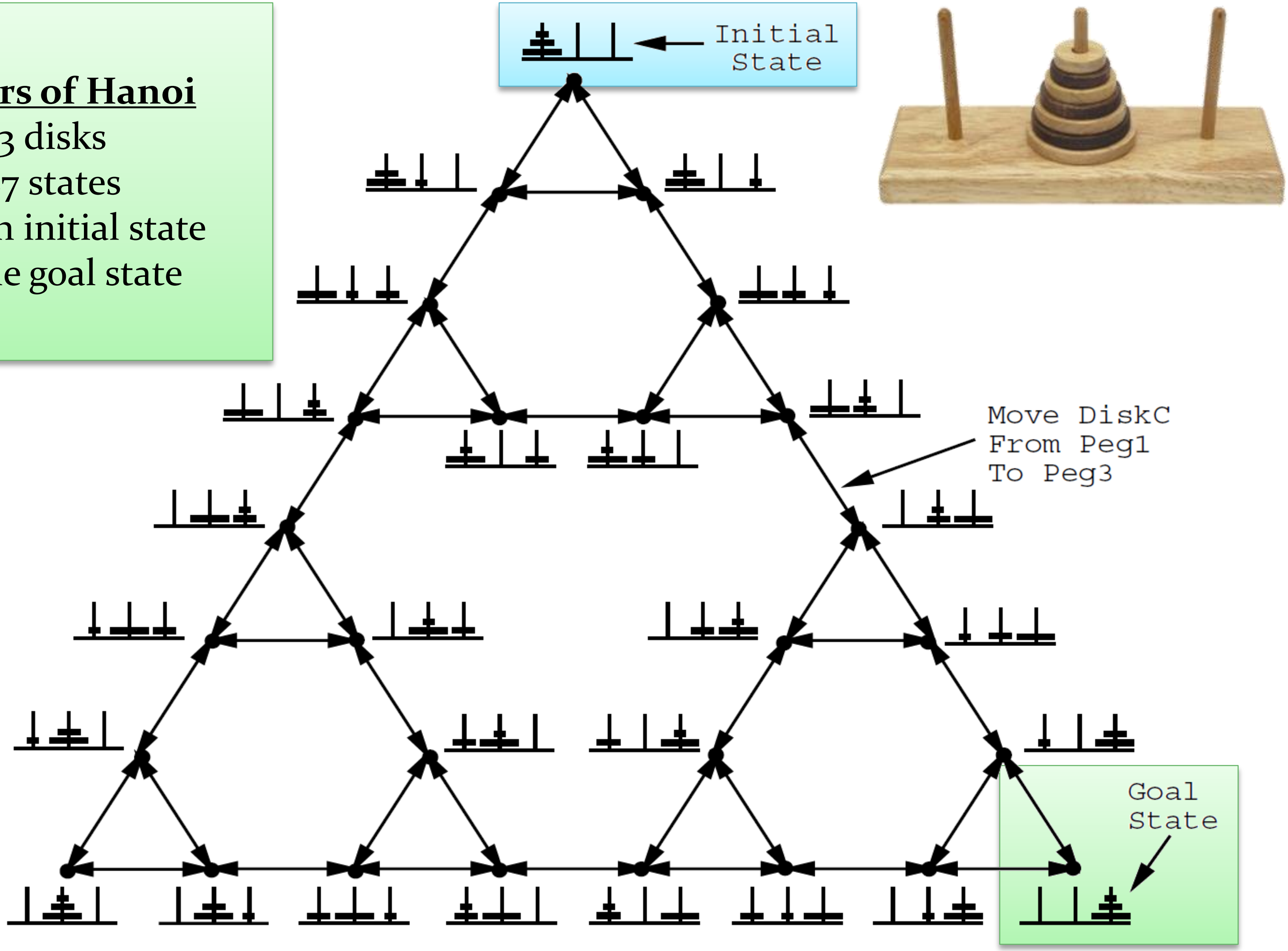
- We can now formally define the **classical planning problem**
  - Let $\Sigma = (S,A,\gamma)$ be a state transition system satisfying the assumptions A0 to A7 (called a **restricted** state transition system in the book)
  - Let $s_0 \in S$ be the **initial state**
  - Let $S_g \subseteq S$ be the **set of goal states**

  - Then, find a **sequence** of **transitions** labeled with actions $[a_1, a_2, \ldots, a_n]$ that can be applied starting at s_0 resulting in a **sequence** of **states** $[s_1, s_2, \ldots, s_n]$ such that $s_n \in S_g$



Example

<u>Towers of Hanoi</u>
3 disks
27 states
Known initial state
Single goal state



Overview – so far

Real World

Abstraction
Approximation
Simplification

Formal Model:
Restricted State Transition System
$\Sigma = (S, A, \gamma)$

Tells us: How the world works
(Only those aspects
that we *need* in our model
in order to solve
interesting problems!)

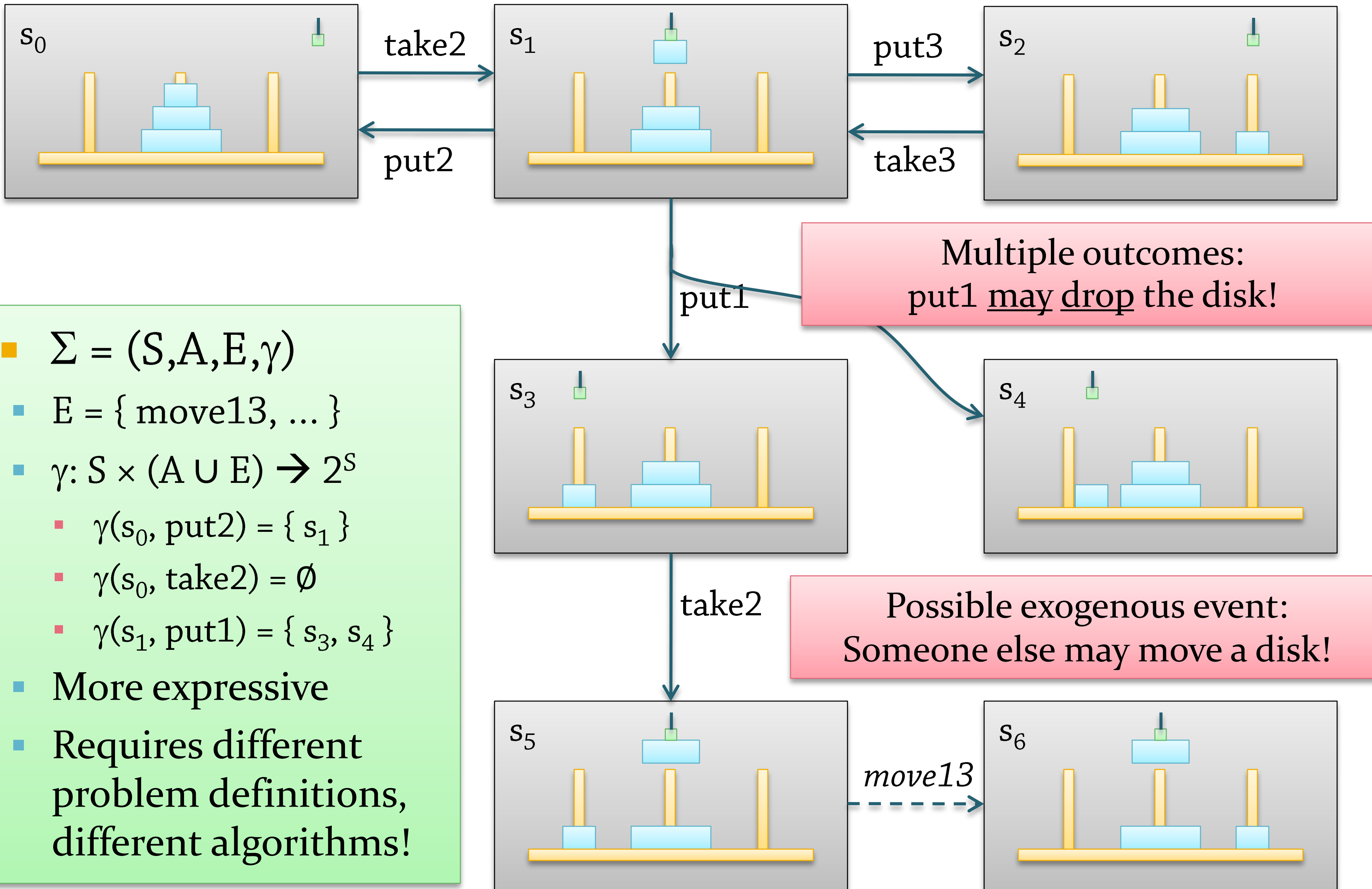
Real World
+ current
problem

Abstraction
Approximation
Simplification

Planning Problem
$\mathcal{P} = (\Sigma, s_0, S_g)$

Tells us:
Which specific problem to solve

Alternative State Transition System (book)



- $\Sigma = (S,A,E,\gamma)$
  - $E = \{$ move13, … $\}$
  - $\gamma: S \times (A \cup E) \rightarrow 2^S$
    - $\gamma(s_0, \text{put2}) = \{ s_1 \}$
    - $\gamma(s_0, \text{take2}) = \emptyset$
    - $\gamma(s_1, \text{put1}) = \{ s_3, s_4 \}$
  - More expressive
  - Requires different problem definitions, different algorithms!

Common Properties

Now we have…

A **formal model**
Capturing the essential aspects
of **classical** planning domains,
instances and plans:
Restricted State Transition System

Quite simple in some respects, but still useful

Many concepts developed here remain valid
in more expressive forms of planning

Can be used to learn about problem structure,
what is difficult and what is easy, etc.

Other types of planning will be considered later!

Common Properties

We still need to define…

(A **formal model**)

A **representation language** allowing you to *conveniently* describe a model

One or more **planners** taking a **specification** in the representation language and generating a **plan** satisfying the goals according to the semantics of the formal model