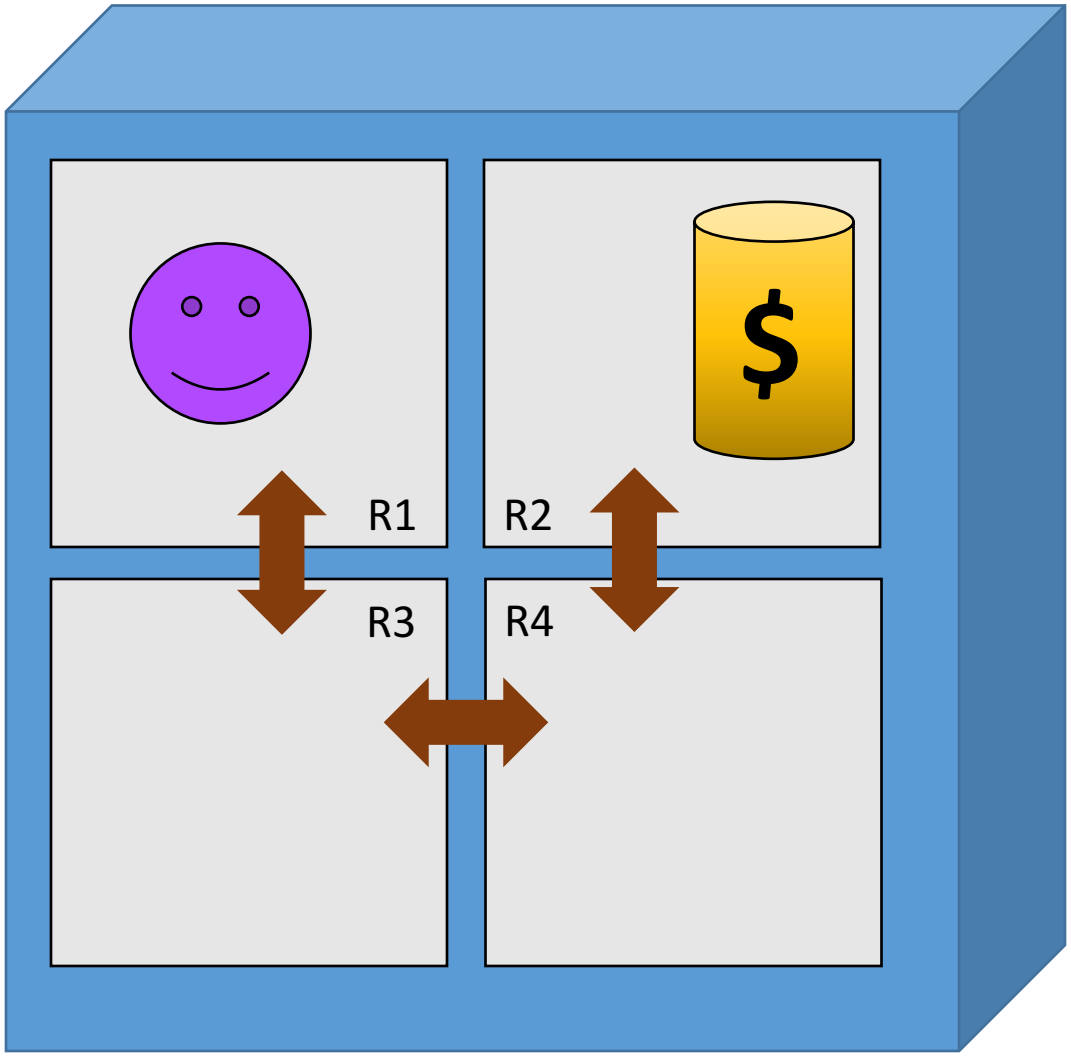
# Planning
# Software Demonstrations

CSC 2542 / Algorithms for Sequential Decision Making

# Agenda

- PDDL
  - (Very) basic PDDL w/ a toy-problem
- Planning systems
  - What's available / How to pick?
  - Demonstrations / How to use?
- Domains
  - Where to find domains
- Other tools / tips

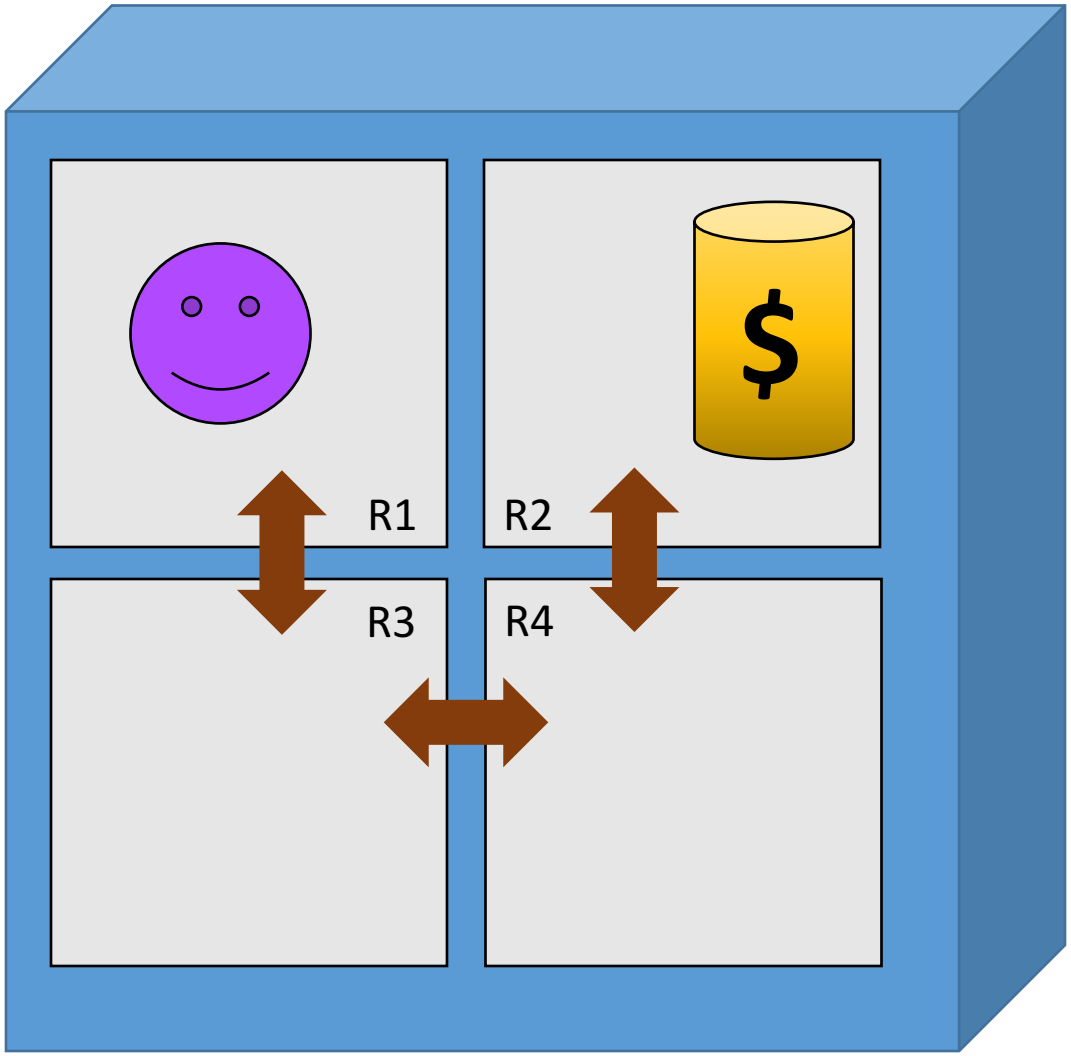
# Toy Problem




Rules:

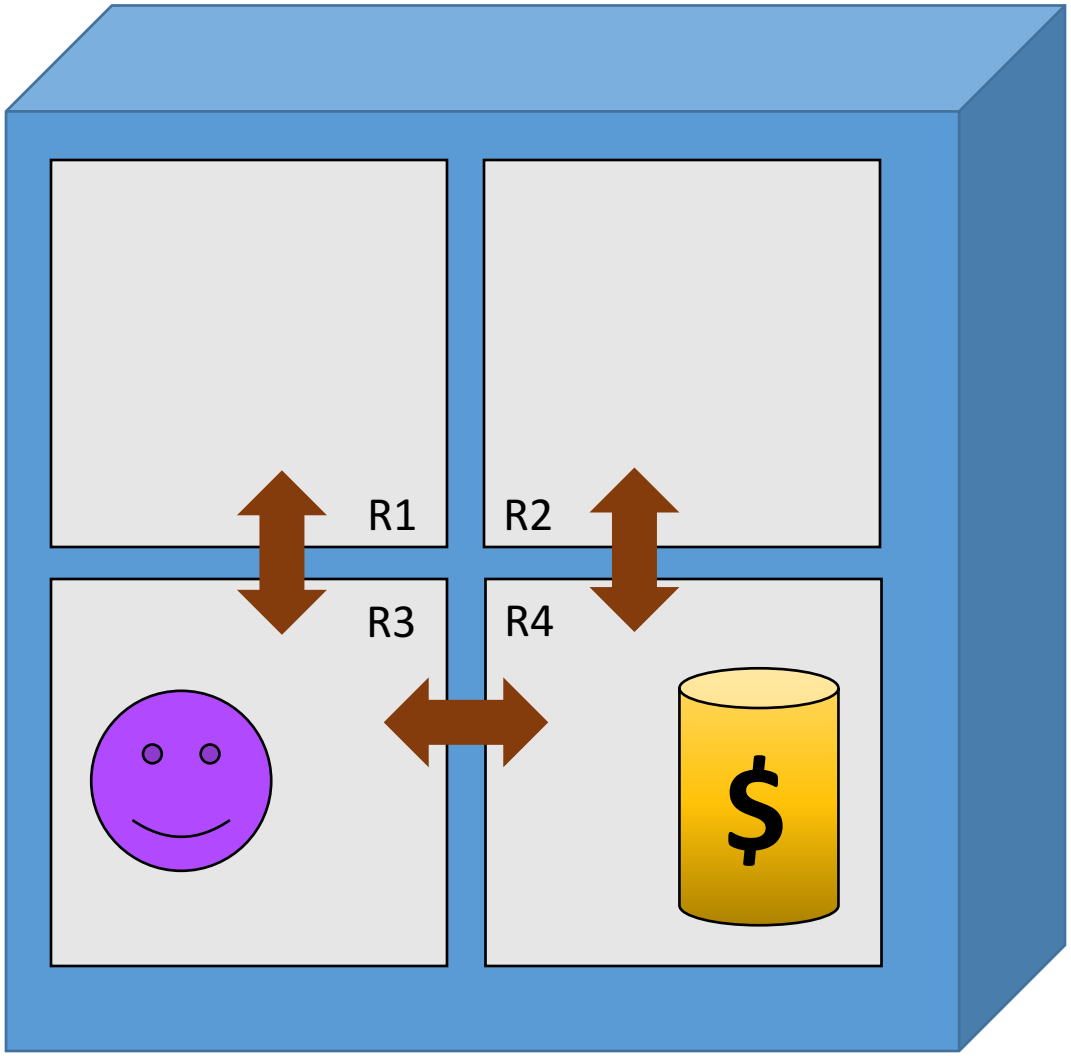
- The person can only move between rooms that are adjacent (have door)
- The person can pick up gold only if in the same room
- Goal: Collect gold and get back to R1

# Toy Problem - PDDL

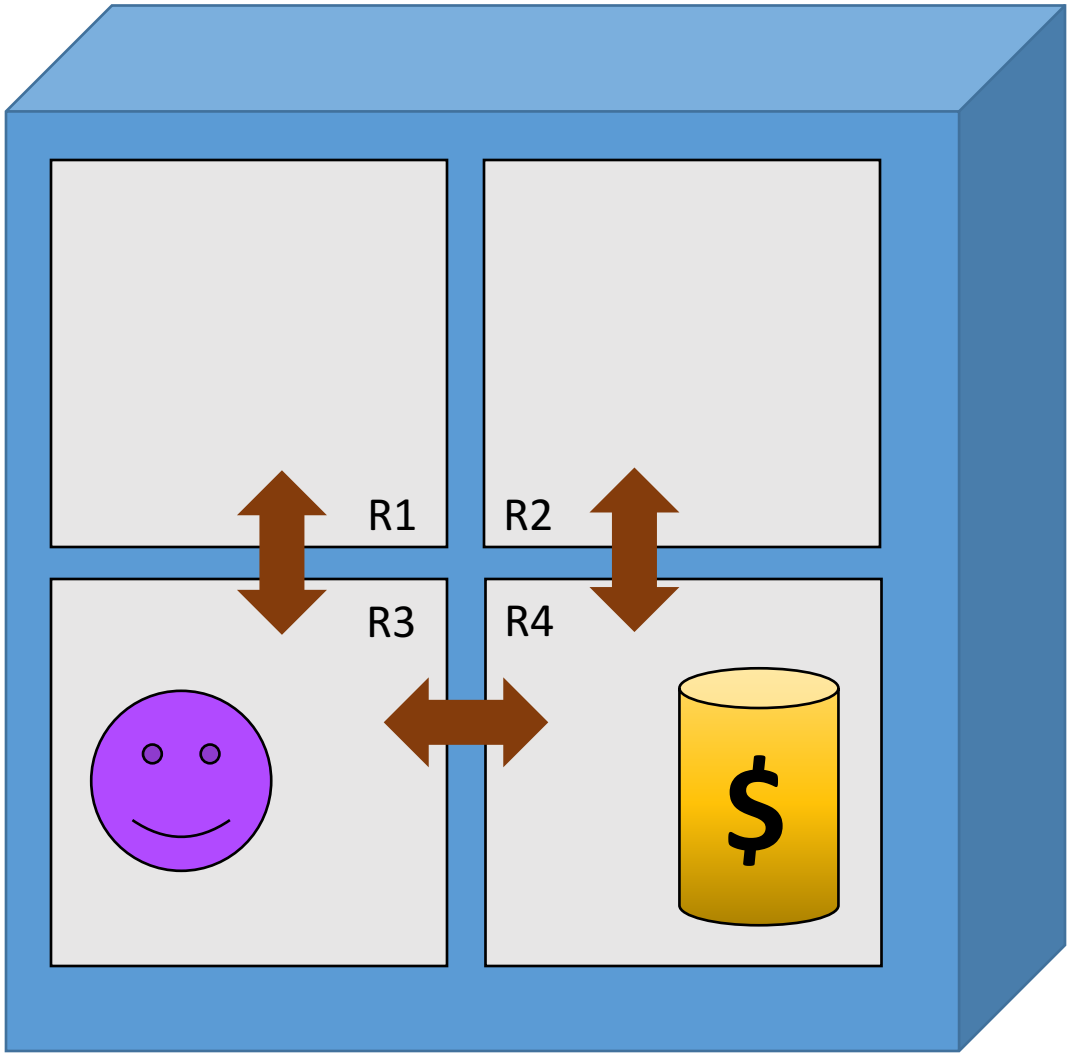


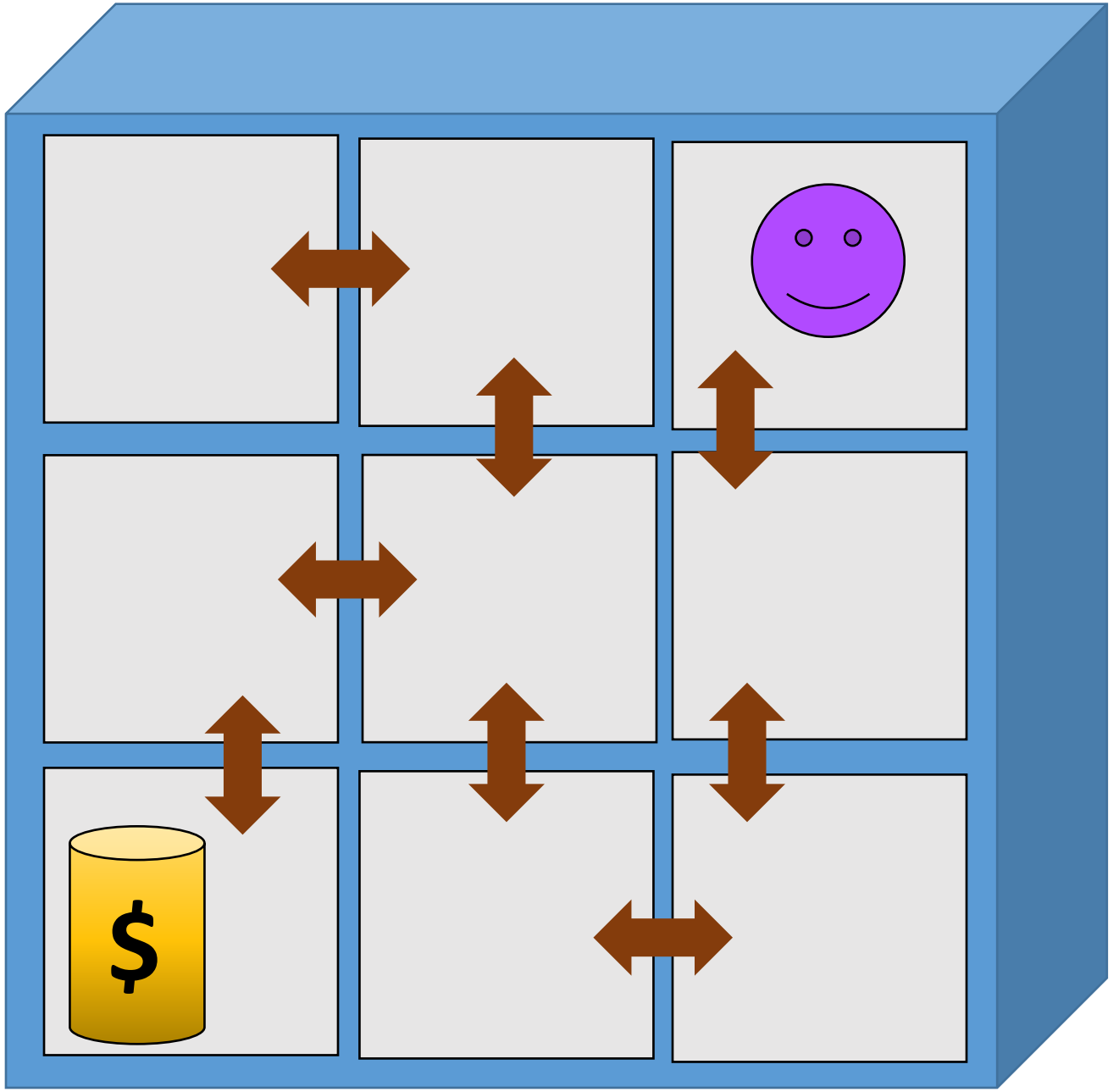

- Objects (Things we care about in our world):
  - Rooms: R1, R2, R3, R4
  - Person P1, Gold G1
- Predicate (Boolean properties of the objects):
  - at(what, where): object what in where
  - adj(room1, room2): room1 is adjacent to room2
  - have(who, what): object who has object what
- Initial state:
  - at(p1,r1), at(g1,r2)
  - adj(R1,R3), adj(R3,R1), adj(R3, R4)
- Goal State:
  - have(R1,G1)
  - at(P1,R1)
- Actions (Ways of changing the world):
  - move (who, from, to), e.g. move(p1,r1,r3)
  - pick(who, what, where) e.g. pick(P1,G1,R1)

# Other Instances

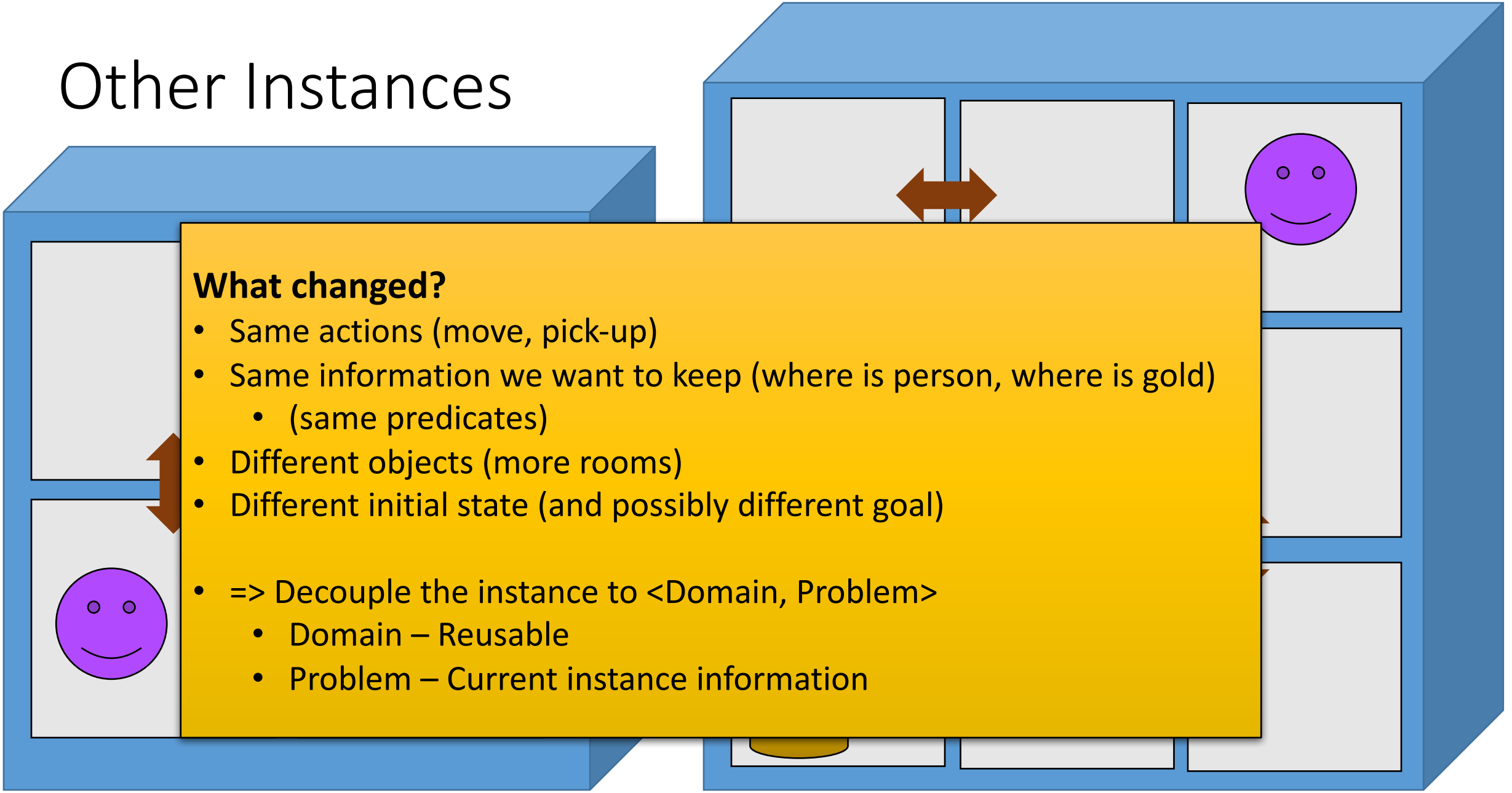


# Other Instances



# Other Instances



**What changed?**

- Same actions (move, pick-up)
- Same information we want to keep (where is person, where is gold)
  - (same predicates)
- Different objects (more rooms)
- Different initial state (and possibly different goal)

- => Decouple the instance to <Domain, Problem>
  - Domain – Reusable
  - Problem – Current instance information

# Modeling in PDDL

- Two components: Domain and Problem(s)

- Domain
  - Predicates
  - Actions

- Problem
  - Objects
  - Initial state
  - Goal state

# Domain and Problem

```
; Domain.pddl
(define (domain DOMAIN_NAME)
  (:requirements [:strips] [:equality] [:typing] [:adl])
  (:predicates (PREDICATE_1_NAME ?A1 ?A2 ... ?AN)
               (PREDICATE_2_NAME ?A1 ?A2 ... ?AN)
               ...)
  (:action ACTION_1_NAME
    [:parameters (?P1 ?P2 ... ?PN)]
    [:precondition PRECOND_FORMULA]
    [:effect EFFECT_FORMULA]
  )
  (:action ACTION_2_NAME ...)
  ...
)
```

```
; problem.pddl
(define (problem PROBLEM_NAME)
  (:domain DOMAIN_NAME)
  (:objects OBJ1 OBJ2 ... OBJ_N)
  (:init ATOM1 ATOM2 ... ATOM_N)
  (:goal CONDITION_FORMULA)
)
```

# Modeling Toy Problem in PDDL

```
(define (domain find-gold)
  (:requirements :strips)

  (:predicates  (at ?what ?room)
                (adj ?room-1 ?room-2)
                (have ?who ?what) )

  (:action walk
    :parameters (?who ?from ?to)
    :precondition (and (at ?who ?from)
                       (adj ?from ?to))
    :effect (and (not (at ?who ?from))
                 (at ?who ?to)) )

  (:action pick-gold
    :parameters (?who ?what ?where)
    :precondition (and (at ?who ?where)
                       (at ?what ?where))
    :effect (and (have ?who ?what)
            (not (at ?what ?where))) )
)
```

```
(define (problem find-gold-problem1)
  (:domain find-gold)

  (:objects r1 r2 r3 r4 g1 p1)

  (:init (adj r1 r3) (adj r3 r1)
         (adj r3 r4) (adj r4 r3)
         (adj r4 r2) (adj r2 r4)
         (at g1 r2)  (at p1 r1)
  )

  (:goal (and (have p1 g1)
              (at p1 r1)))
)
```

# Using a planner to solve the problem

- http://planning.domains/
  - Online service providing “Solver on the Cloud”
    - No need to compile, install, and run a local planner
    - Problems can be posted to solver as JSON over HTTP
      - Easy from Python / JS
  - Online PDDL editor
    - Can type PDDL on website and run planner
  - [Demonstration]

# Modeling Toy Problem in PDDL – 1st Attempt

```
(define (domain find-gold)
  (:requirements :strips)

  (:pred                                                    roblem1)


  (:acti                                                     l)
    :par                                                     r1)
    :pre                                                     r3)
                                                             r4)
    :eff                                                    r1)


  (:acti
    :par
    :pre

    :eff
         (not (at ?what ?where))) )
)
```

Proposed Plan:

Move G1, R2, R4
Move G1, R4, R3
Move G1, R3, R1
PickUp P1, G1, R1

Is it good?

```
(define (domain find-gold)
  (:requirements :strips)

  (:predicates   (at ?what ?room)
                 (adj ?room-1 ?room-2)
                 (have ?who ?what)
                 (is-room ?who)
                 (is-person ?who)
                 (is-gold ?what))
```

Revise PDDL - Gold can't walk

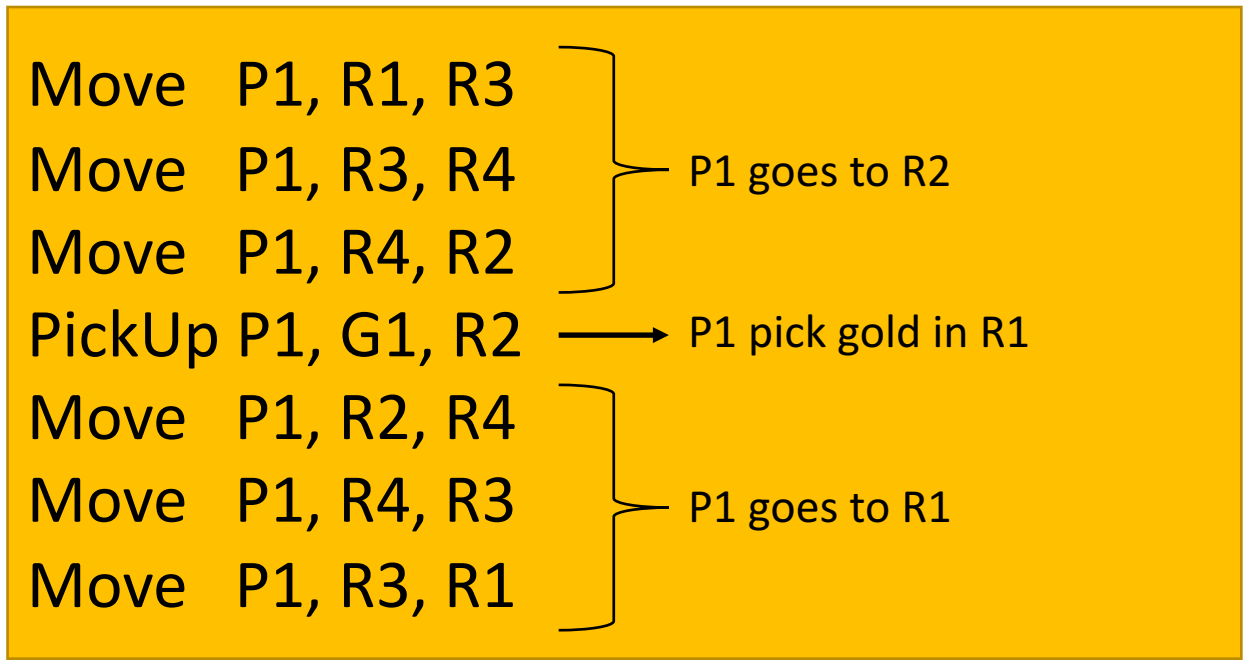
- Add "is_x" predicates to verify the object we apply actions on

- Run planner again

```
(:action move
   :parameters (?who ?from ?to)
   :precondition (and (is-person ?who)
                      (is-room ?from)
                      (is-room ?to)
                      (at ?who ?from)
                      (adj ?from ?to))
   :effect (and (not (at ?who ?from))
                (at ?who ?to)) )

 (:action pick-gold
   :parameters (?who ?what ?where)
   :precondition (and (is-person ?who)
                      (is-gold ?what)
                      (is-room ?where)
                      (at ?who ?where)
                      (at ?what ?where))
   :effect (and (have ?who ?what)
                (not (at ?what ?where))) )
)
```

```
(define (domain find-gold)
  (:requirements :strips)

  (:predicates   (at ?what ?room)
                 (adj ?room-1 ?room-2)
                 (have ?who ?what)
                 (is-room ?who)
                 (is-person ?who)
                 (is-gold ?what))
```

```
(:action move
   :parameters (?who ?from ?to)
   :precondition (and (is-person ?who)
                      (is-room ?from)
                      (is-room ?to)
                      (at ?who ?from)
                      (adj ?from ?to))
   :effect (and (not (at ?who ?from))
                (at ?who ?to)) )

 (:action pick-gold
   :parameters (?who ?what ?where)
   :precondition (and (is-person ?who)
                      (is-gold ?what)
                      (is-room ?where)
                      (at ?who ?where)
                      (at ?what ?where))
   :effect (and (have ?who ?what)
                (not (at ?what ?where))) )
)
```



# Planners

- Fast Forward
  - https://fai.cs.uni-saarland.de/hoffmann/ff.html
- Fast Downward
  - http://www.fast-downward.org
- Planner on the cloud
  - http://planning.domains/
- Madagascar (SAT-based planner)
  - https://users.ics.aalto.fi/rintanen/satplan.html

# Finding the right planner

- IPC – International Planning Competition
  - https://helios.hud.ac.uk/scommv/IPC-14/

- Different Tracks
  - Deterministic Track
  - Learning Track
  - Uncertainty / Probabilistic
  - Unsolvability

- Different Goals:
  - Solution Quality
  - Time to Solution
  - CPU %
  - …

# Domains (and Problems)

- IPC – International Planning Competition
  - http://icaps-conference.org/index.php/main/competitions
  - https://helios.hud.ac.uk/scommv/IPC-14/
- http://planning.domains
  - https://bitbucket.org/planning-researchers/classical-domains
- Fast Downward benchmarks repository
  - https://bitbucket.org/aibasel/downward-benchmarks/src
- Fast Forward domains
  - https://fai.cs.uni-saarland.de/hoffmann/ff-domains.html

# Plan Validation using VAL

- VAL - The Automatic Validation Tool For PDDL, including PDDL3 and PDDL+
- Validate a plan(s) with the domain and problem files
- If plan is invalid – offer a Plan Repair Advice
- Can produce latex with validation details, graphs, gantt charts, etc.

- Usage:
  - validate [options] domain.pddl problem.pddl plan1.pddl plan2.pddl

- https://github.com/KCL-Planning/VAL
- http://www.inf.kcl.ac.uk/staff/planningtmp/software/val.html

# Software demonstration

- Running IPC planner w/ IPC domain
- Validating plan
- IPC domains on planning.domains

# PDDL Tools

- IDEs - syntax highlights, etc
  - MyPDDL – for Sublime Text
    - http://pold87.github.io/myPDDL/
  - PDDL mode for Emacs
    - https://www.emacswiki.org/emacs/pddl-mode.el

- Planning systems development
  - Pyperplan (https://bitbucket.org/malte/pyperplan) - Python
  - PDDL4J (https://github.com/pellierd/pddl4j) - Java
  - LAPKT (http://lapkt.org) - C++/Python

# Other Good General Resources

- Automated Planning Course
  (Jonas Kvarnström, Linköping University)
  - https://www.ida.liu.se/~TDDD48/index.en.shtml

- A Beginner's Introduction to Heuristic Search Planning
  (Malte Helmert and Gabriele Röger, AAAI-15)
  - http://ai.cs.unibas.ch/misc/tutorial_aaai2015/

- Patrik Haslum’s Planning Resources
  - http://users.cecs.anu.edu.au/~patrik/#resources