OMPL: The Open Motion Planning Library

Mark Moll
Department of Computer Science
Rice University
Houston, TX
USA

Intended use

- Education
- Motion planning research
- Industry

Design objectives

- Clarity of concepts
- Efficiency
- Simple integration with other software packages
- Straightforward integration of external contributions

Other motion planning software

- **MPK**, Schwarzer, Saha, Latombe
- **MSL**, LaValle et al.
- **OpenRAVE**, Diankov & Kuffner
- **KineoWorks**, Laumond et al.
- **OOPSMP**, Plaku et al.

Other related robotics software

- ROS
- Player/Stage, Player/Gazebo
- Webots
- MORSE
- Microsoft Robotics Developer Studio

Main features of OMPL

OMPL in a nutshell

- Common core for sampling-based motion planners
- Includes commonly-used heuristics
- Takes care of many low-level details often skipped in corresponding papers

Abstract interface to *all* core motion planning concepts

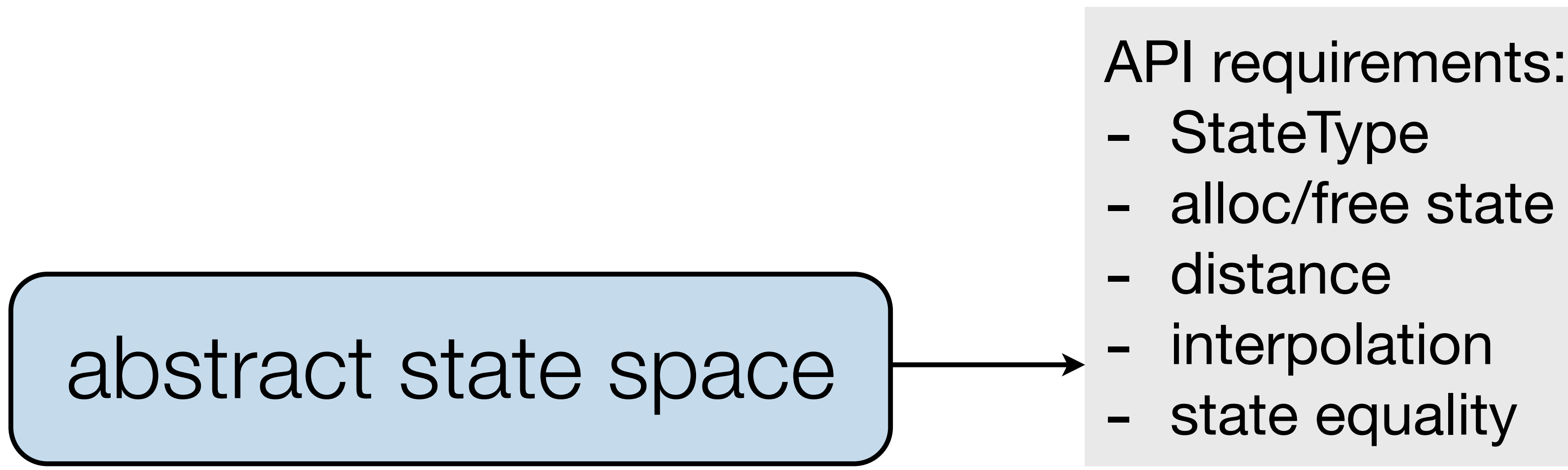
- state space / control space
- state validator (e.g., collision checker)
- sampler
- goal (problem definition)
- planner
- ...

except robot & workspace...

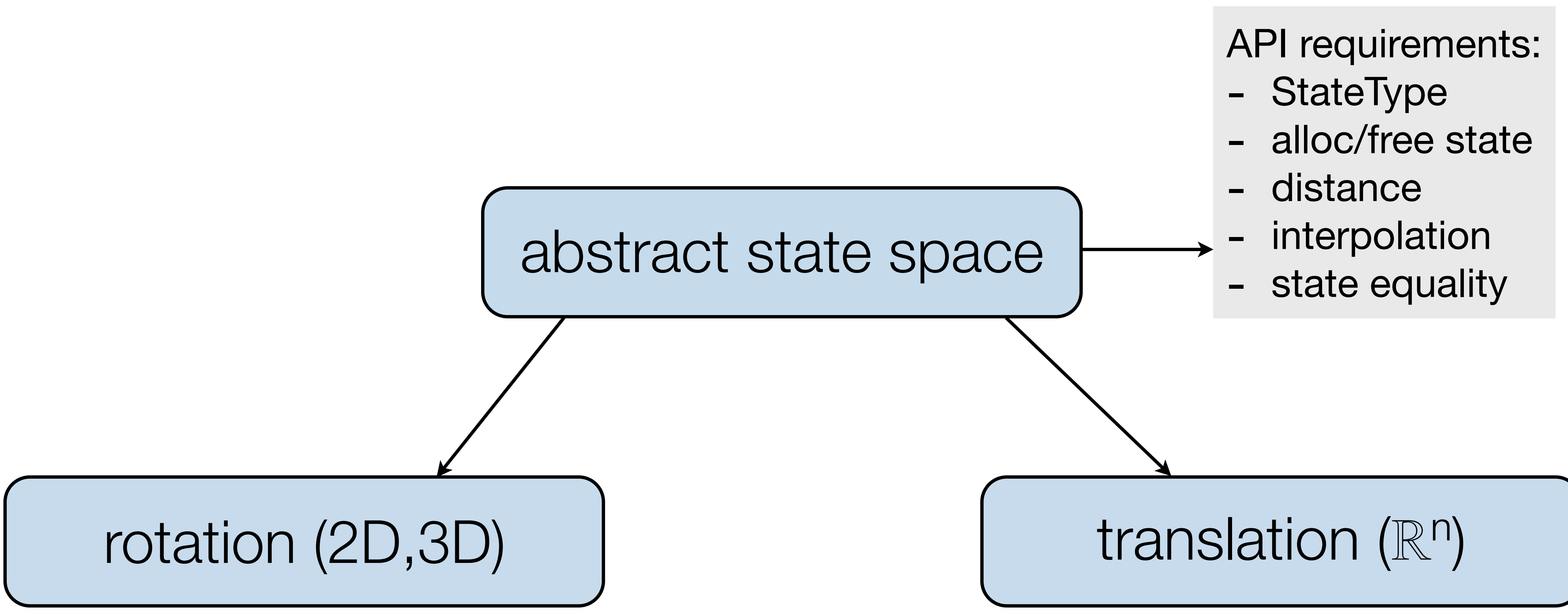
States & state spaces

abstract state space

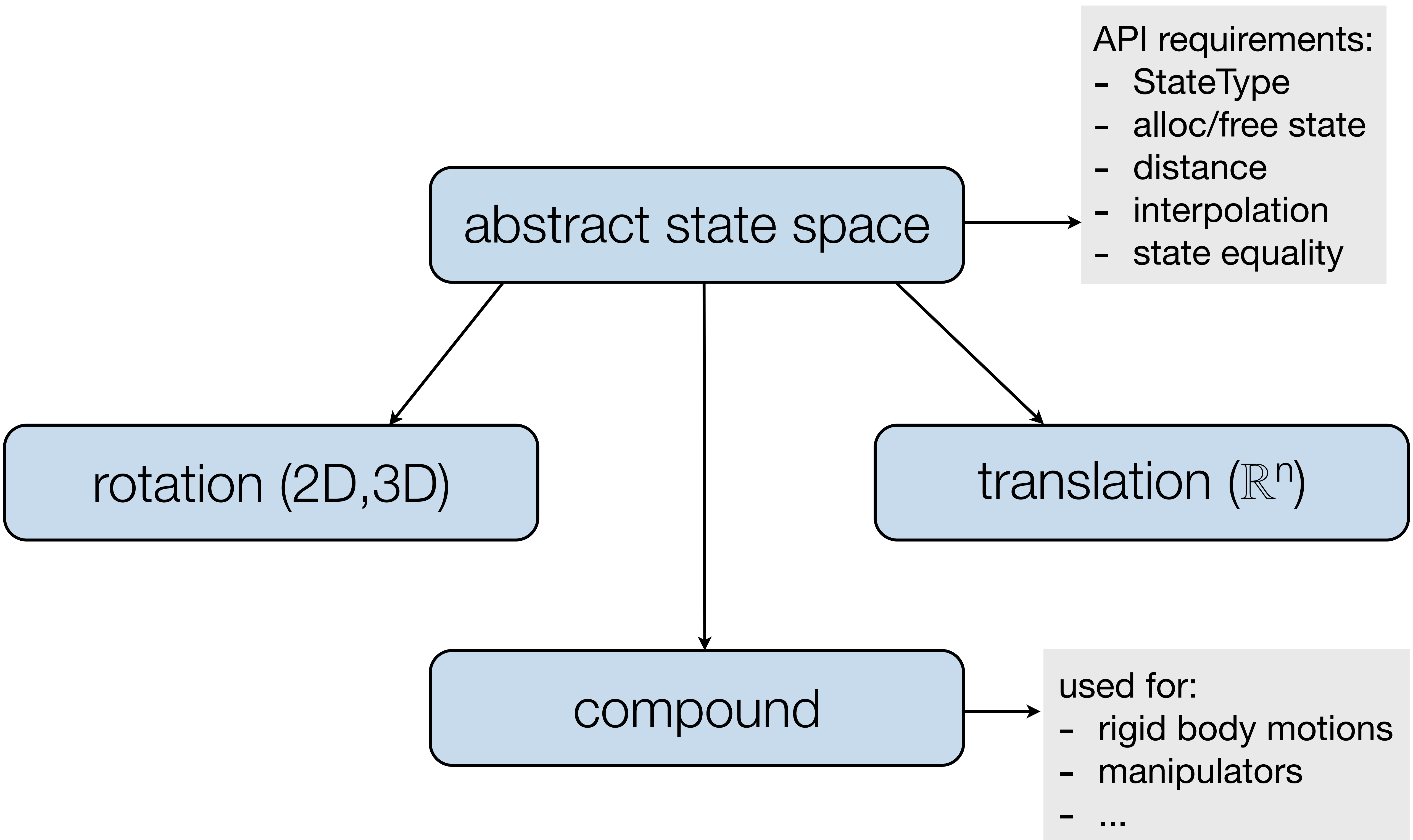
States & state spaces



States & state spaces



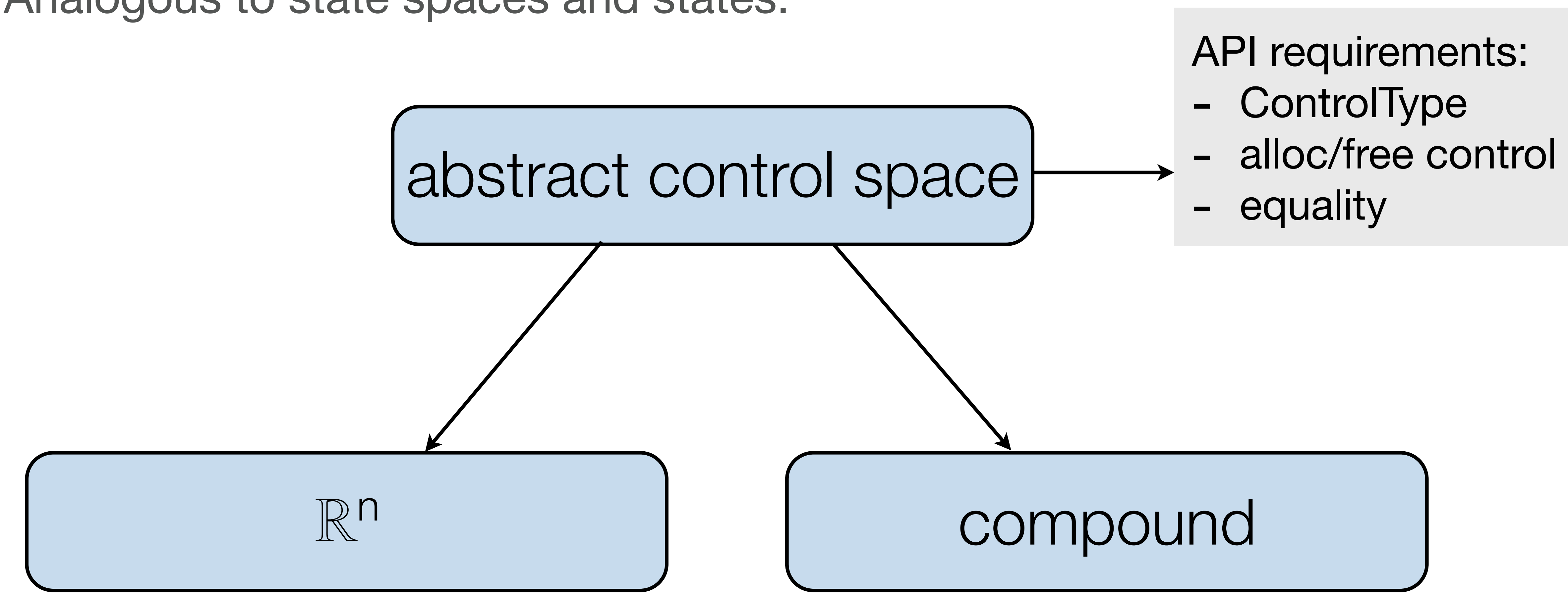
States & state spaces



Control spaces & controls

- Needed only for control-based planning

- Analogous to state spaces and states:



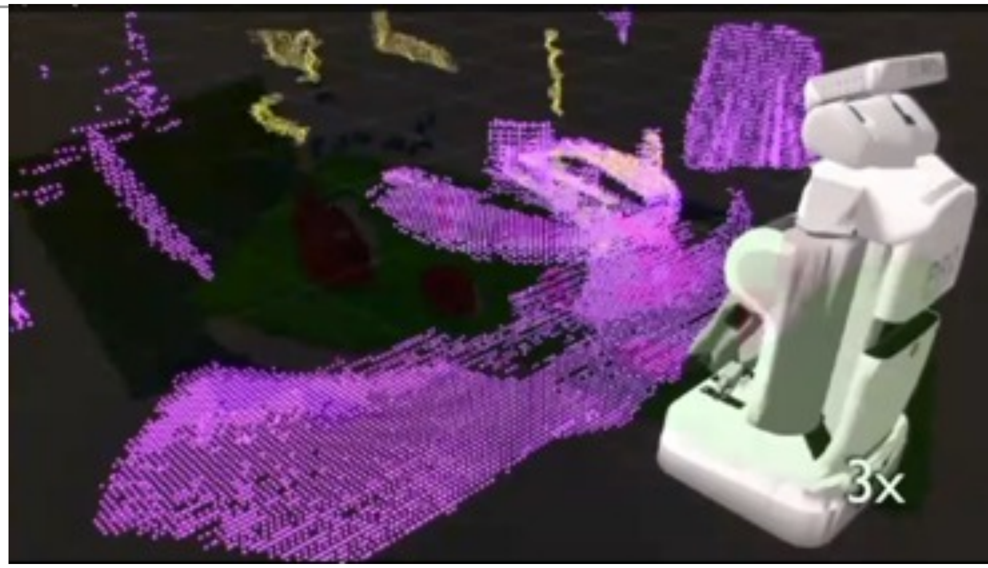
State validators

- Problem-specific; **must** be defined by user **or** defined by layer on top of OMPL core → **ompl_ros_interface**
- Checks whether state is collision-free, joint angles and velocities are within bounds, etc.
- **Optionally,** specific state validator implementations can return
    - distance to nearest invalid state (i.e., nearest obstacle)
    - gradient of distance

  Can be exploited by planners / samplers!

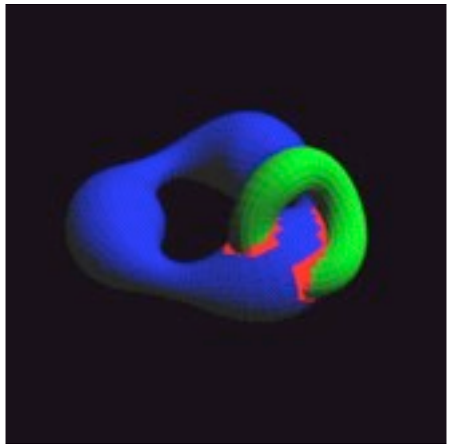
Most common state validator: collision checker

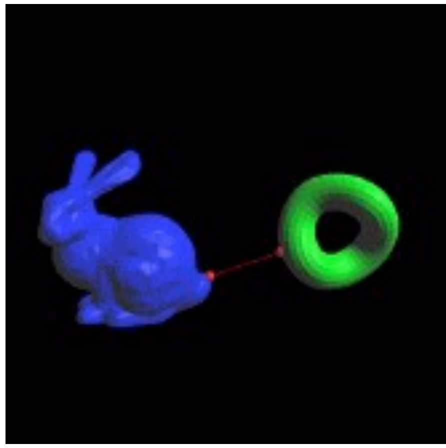
image from pointclouds.org



Several options:

- Implemented in ROS on top of sensor-derived world model
- Implemented in OMPL.app for triangle meshes using PQP library
- Easy to add wrappers for other libraries





images from PQP web site

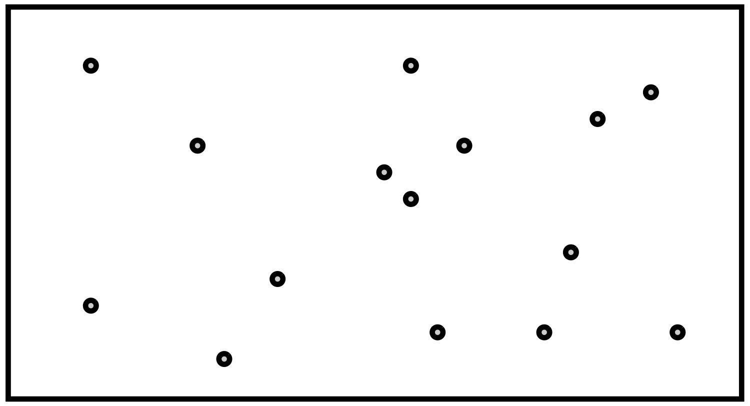
*Need to define **specific** world representation to implement collision checking*

Samplers

- For every **state space** there needs to be a **state sampler**
- State samplers need to support the following:

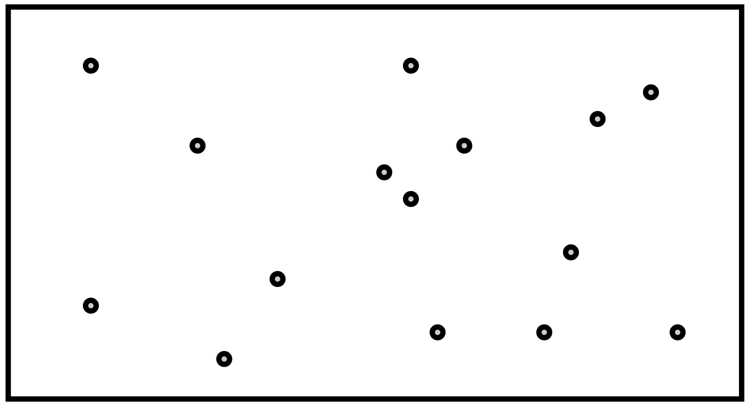
Samplers

- For every **state space** there needs to be a **state sampler**
- State samplers need to support the following:
    - sample uniform

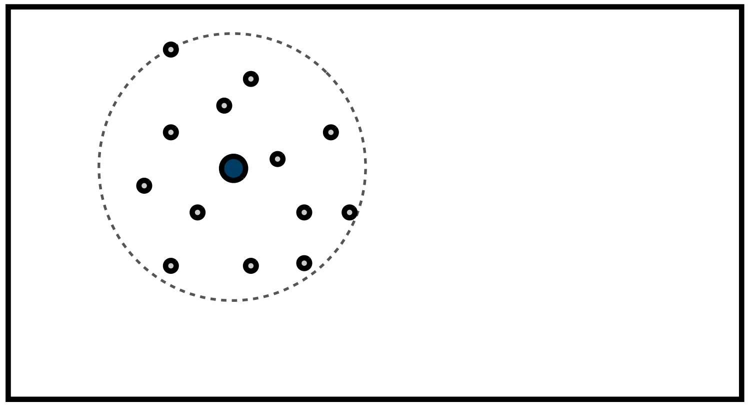


Samplers

- For every **state space** there needs to be a **state sampler**
- State samplers need to support the following:
  - sample uniform

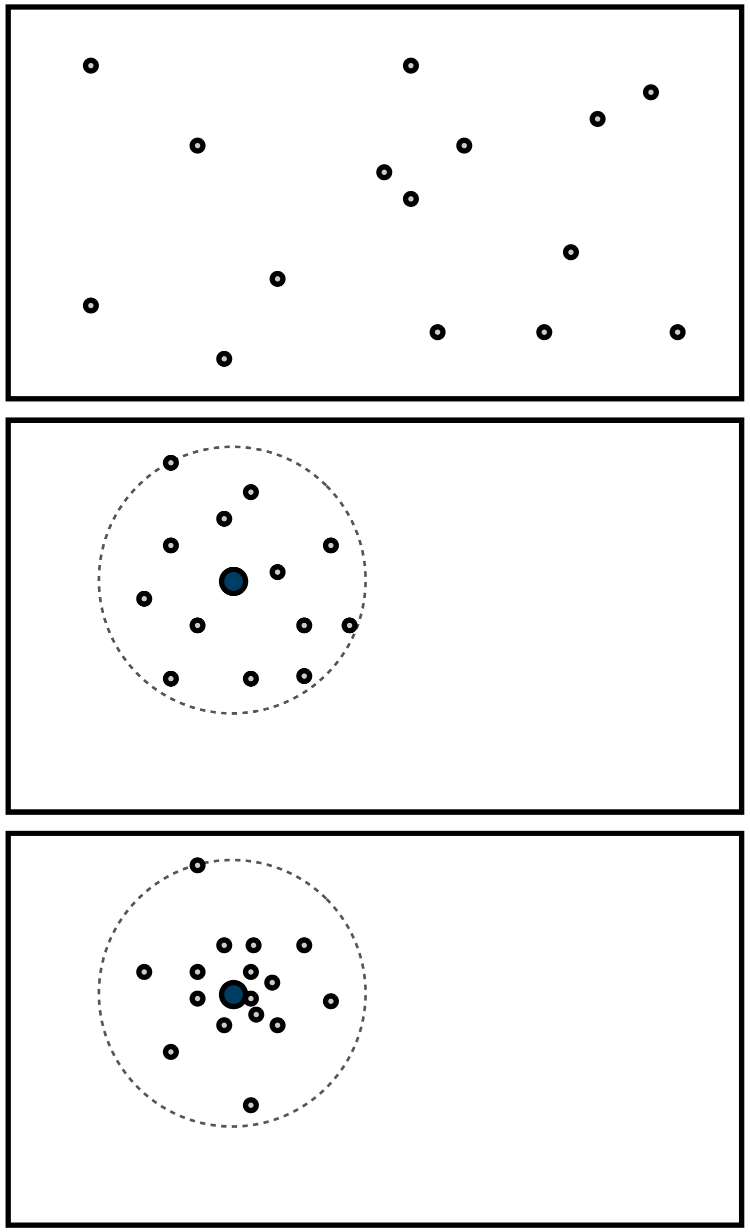


  - sample uniform near given state



Samplers

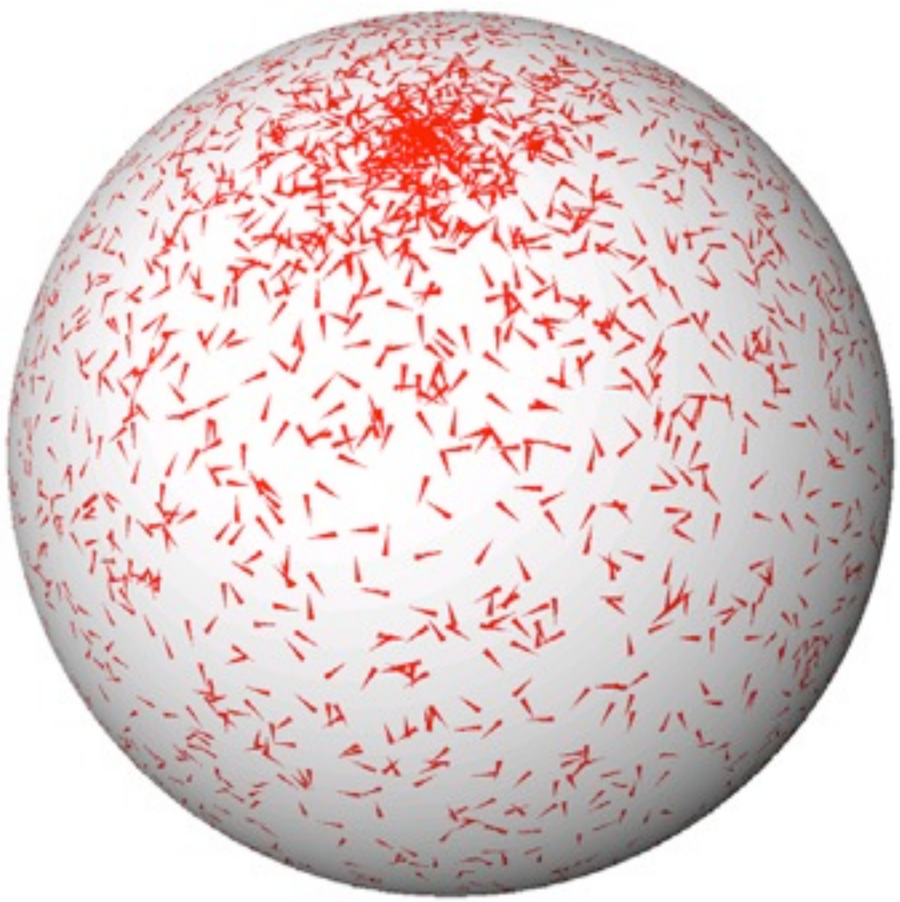
- For every **state space** there needs to be a **state sampler**
- State samplers need to support the following:
  - sample uniform
  - sample uniform near given state
  - sample from Gaussian centered at given state



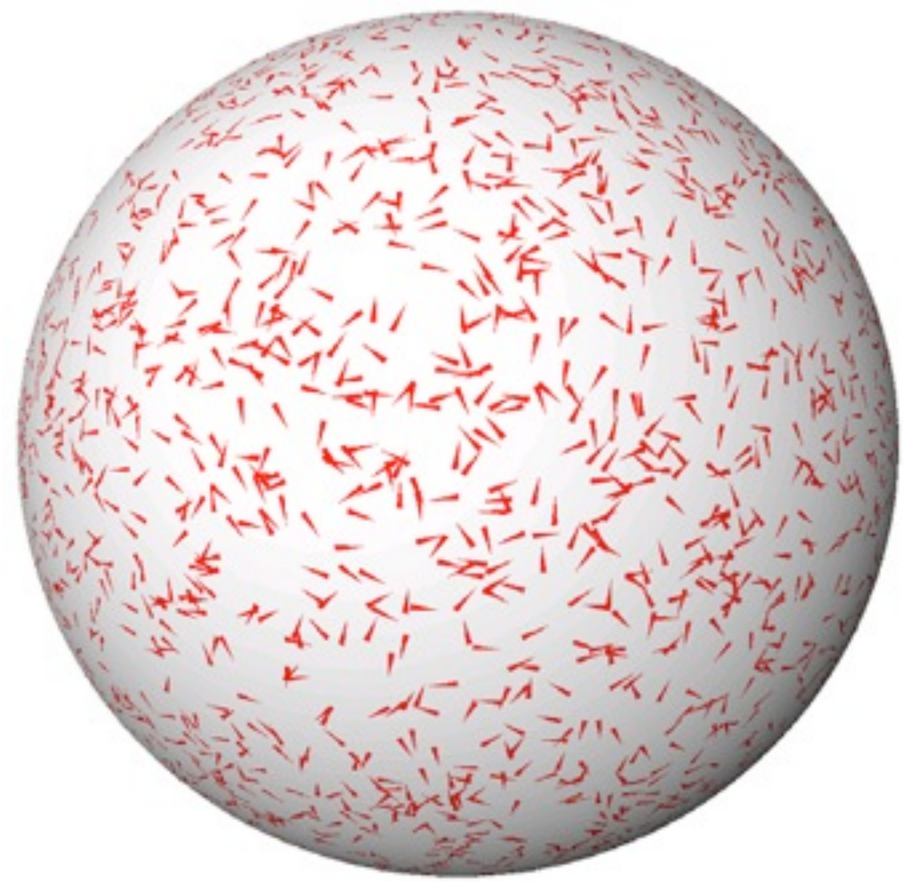
Many ways to get sampling wrong

Example: uniformly sampling 3D orientations

naïve & wrong:



correct:



Images from Kuffner, ICRA '04

Similar issues occur for nearest neighbors

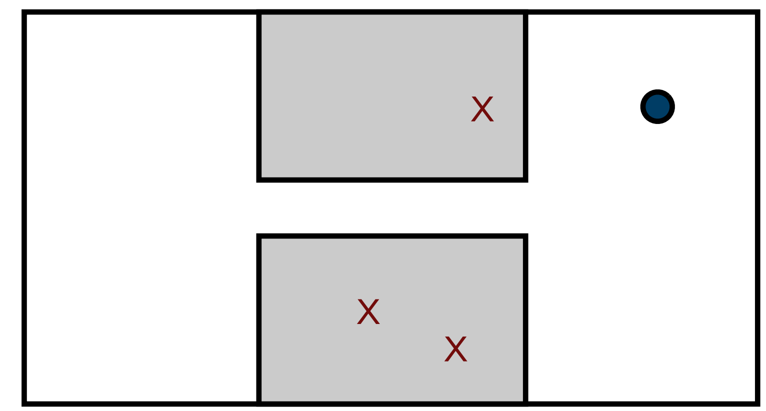
- *k* nearest neighbors can be computed efficiently with *k*d-trees in **low-dimensional, Euclidean** spaces.

- In high-dimensional spaces **approximate** nearest neighbors much better

- In **non-Euclidean** spaces (e.g., any space that includes **rotations**), other data structures are necessary

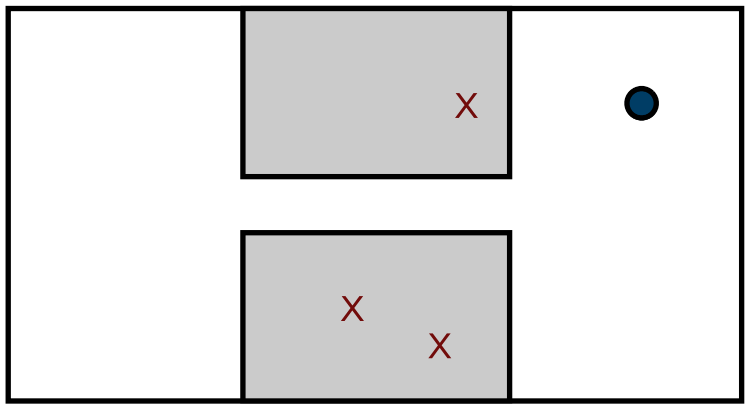
Valid state samplers

- ***Valid* state samplers** combine low-level **state samplers** with the **validity checker**
- Simplest form: sample at most *n* times to get valid state or else return failure



Valid state samplers

- ***Valid* state samplers** combine low-level **state samplers** with the **validity checker**

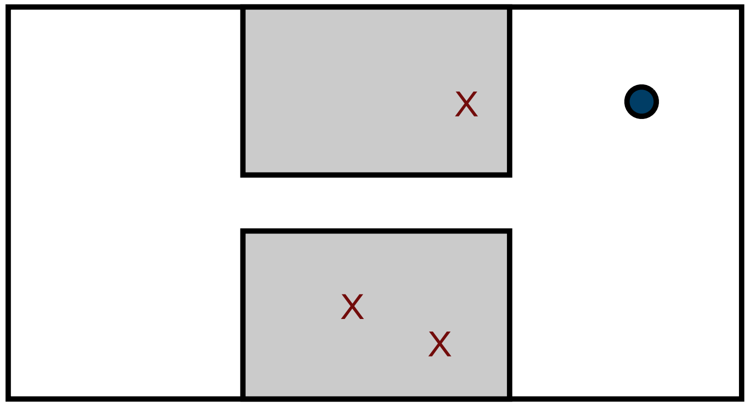


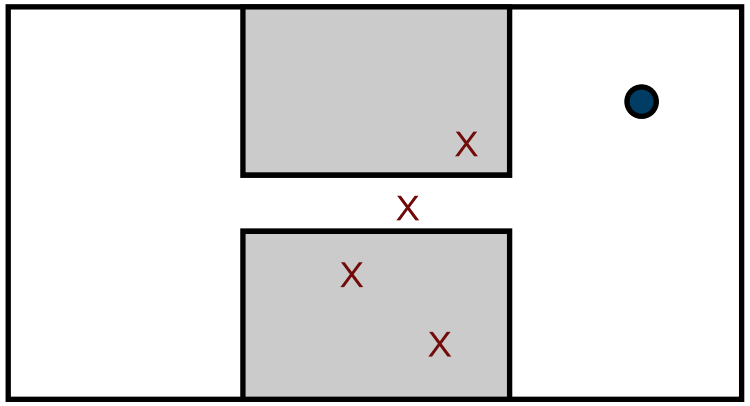
- Simplest form: sample at most *n* times to get valid state or else return failure

- Other sampling strategies:

Valid state samplers

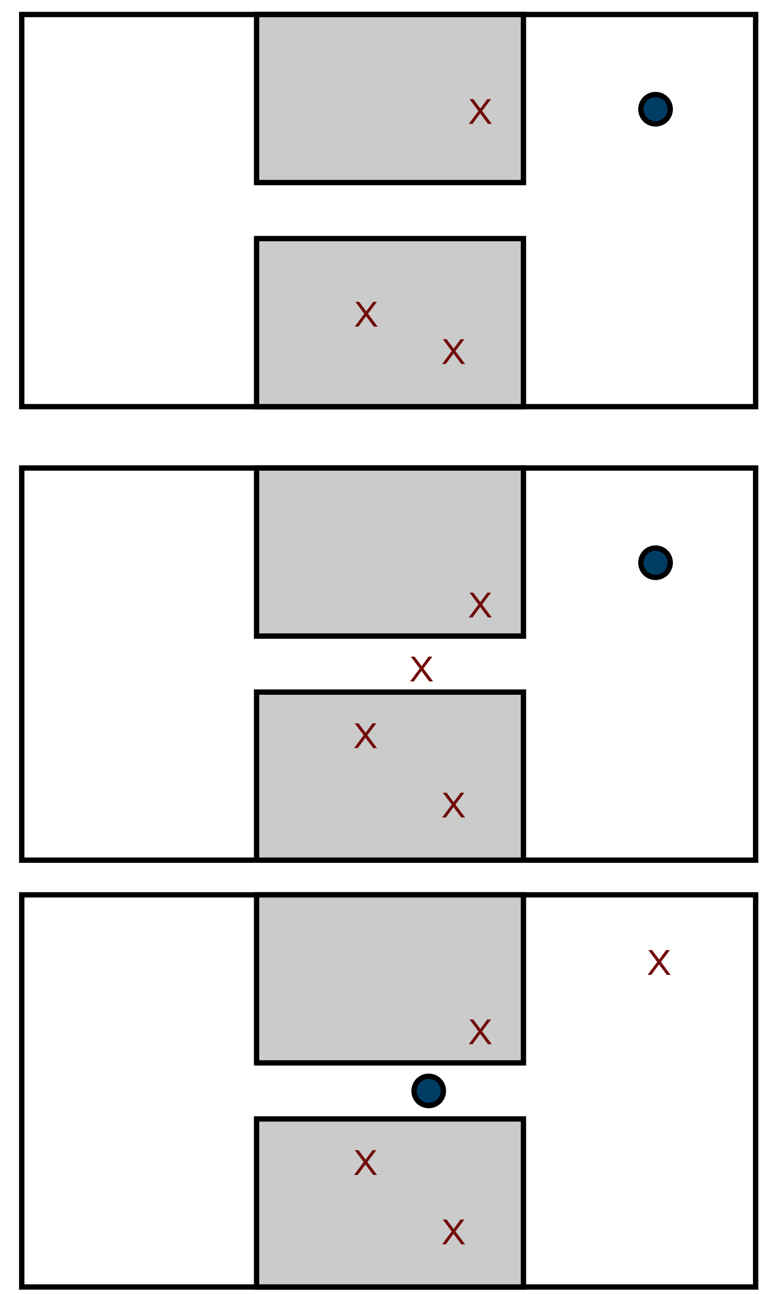
- ***Valid* state samplers** combine low-level **state samplers** with the **validity checker**
- Simplest form: sample at most *n* times to get valid state or else return failure
- Other sampling strategies:
  - Try to find samples with a large clearance



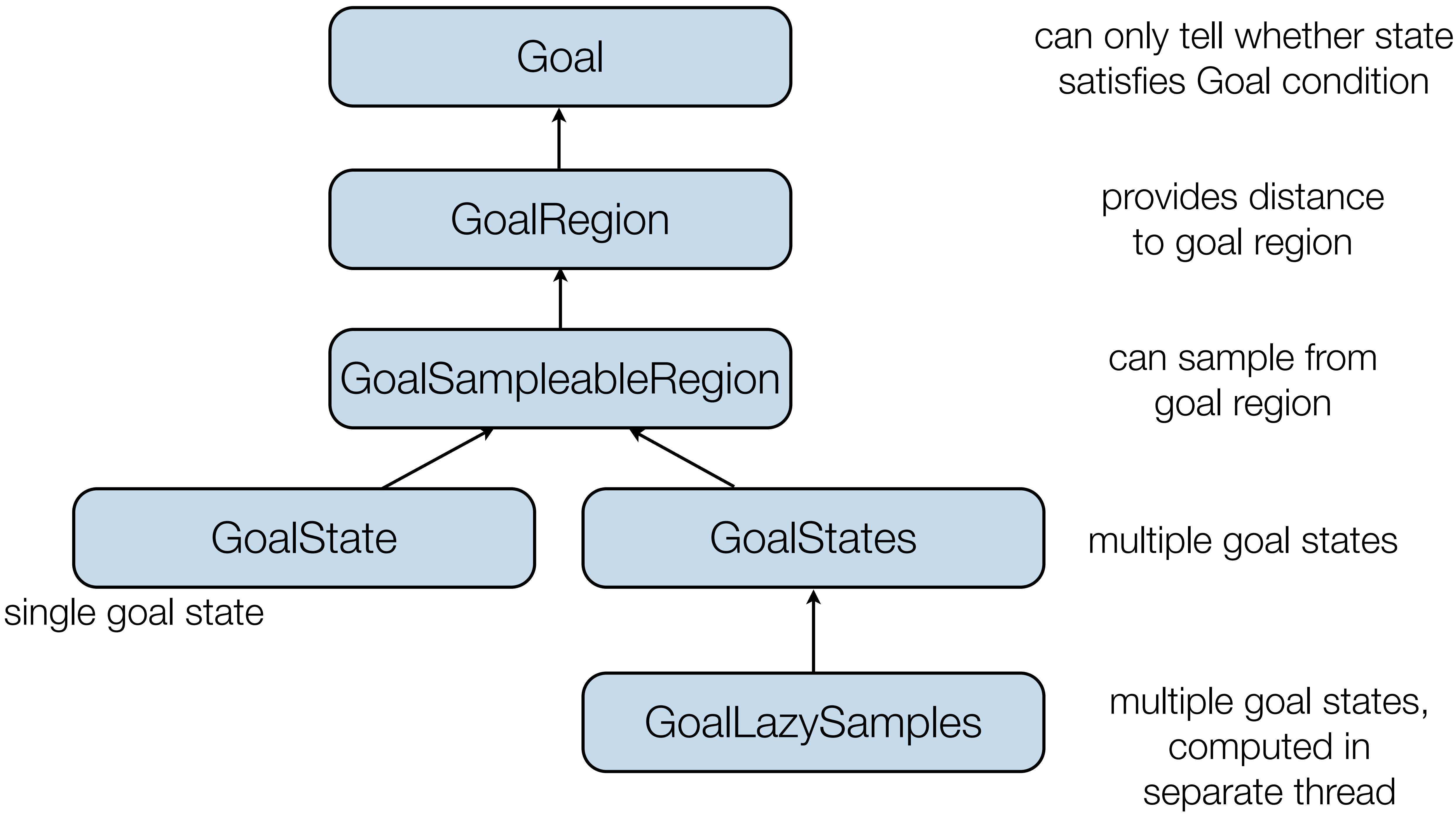


Valid state samplers

- ***Valid* state samplers** combine low-level **state samplers** with the **validity checker**
- Simplest form: sample at most *n* times to get valid state or else return failure
- Other sampling strategies:
  - Try to find samples with a large clearance
  - Try to find samples near obstacles (more dense sampling in/near narrow passages)



Goals


Goal
can only tell whether state
satisfies Goal condition
GoalRegion
provides distance
to goal region
GoalSampleableRegion
can sample from
goal region
GoalState
single goal state
GoalStates
multiple goal states
GoalLazySamples
multiple goal states,
computed in
separate thread

Planners

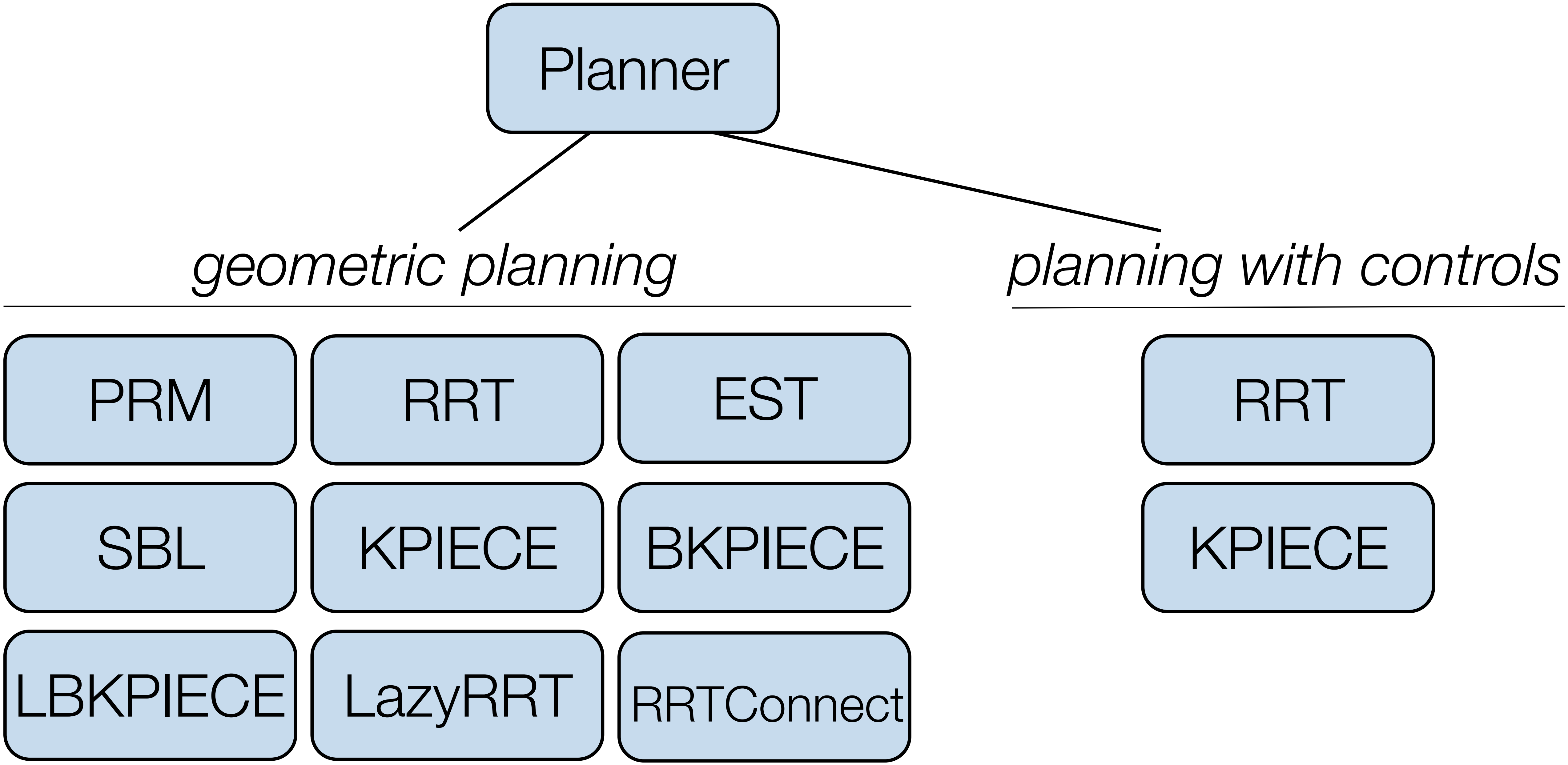
- Take as input a **problem definition**:
  object with one or more **start states** and a **goal object**

- Planners need to implement two methods:

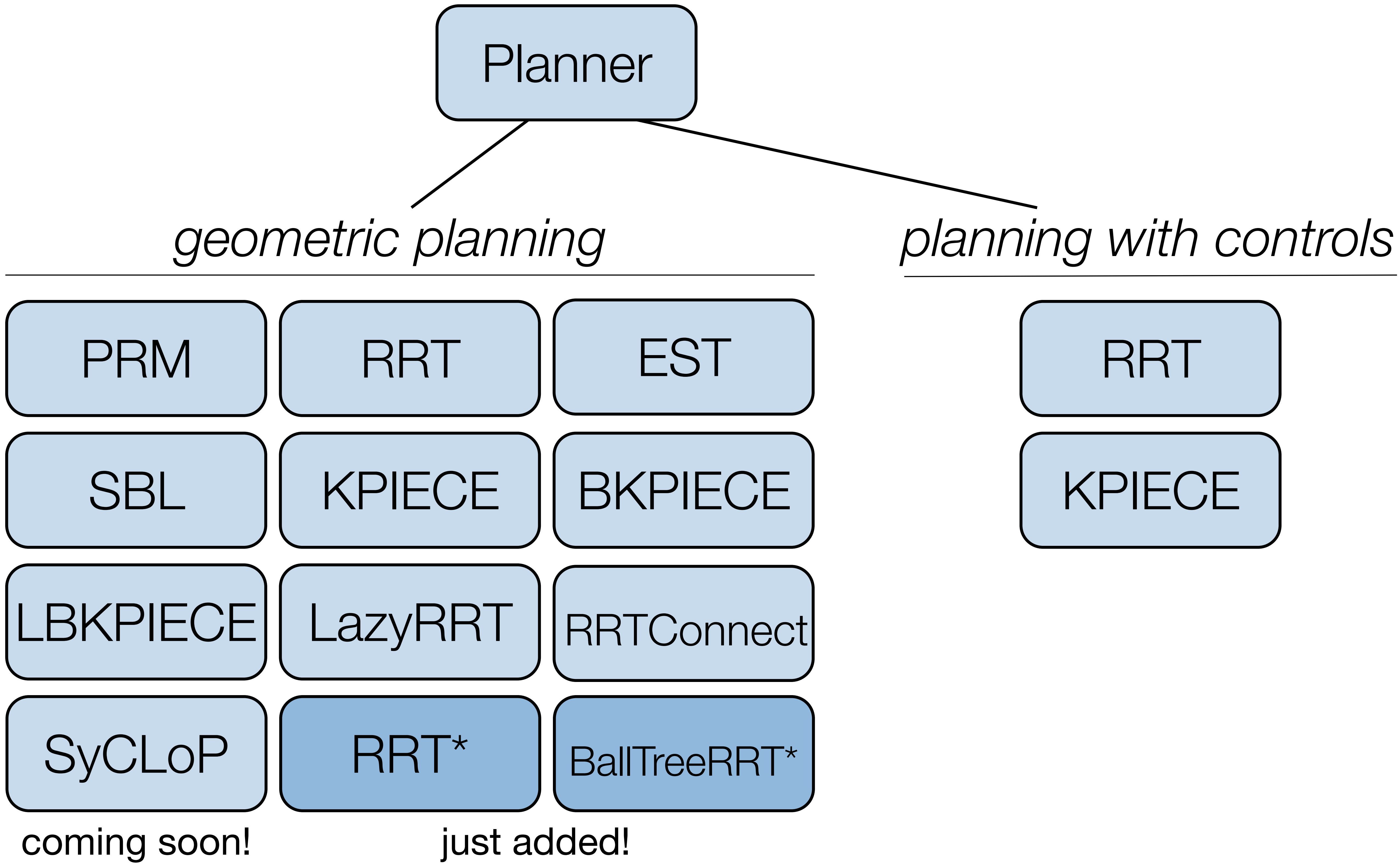
  - **solve:**
    – takes **PlannerTerminationCondition** object as argument
    – termination can be based on timer, external events, ...

  - **clear:**
    clear internal data structures, free memory, ready to run solve again

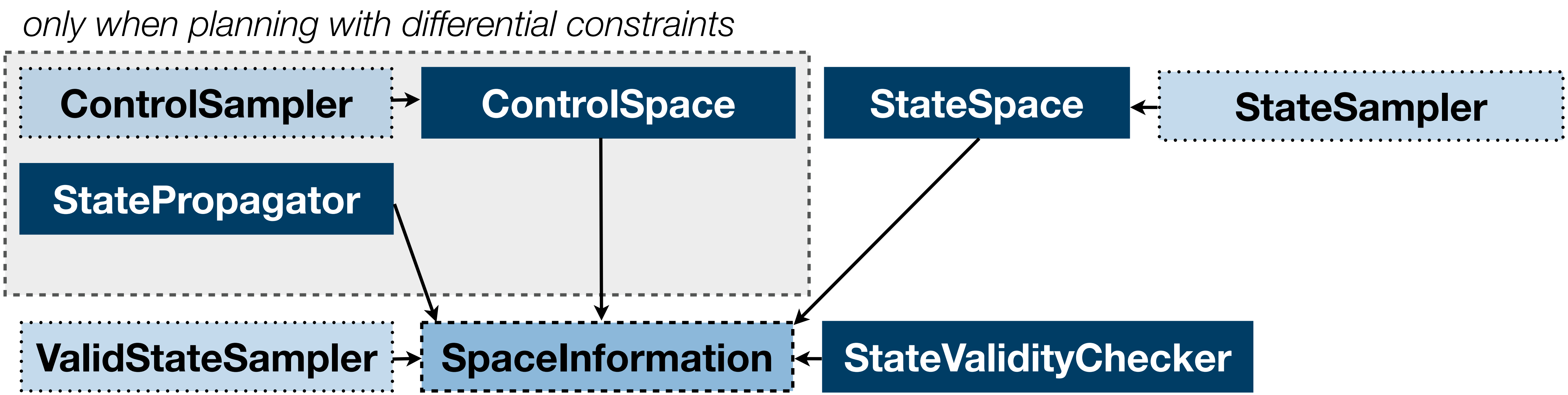
Many planners available in OMPL



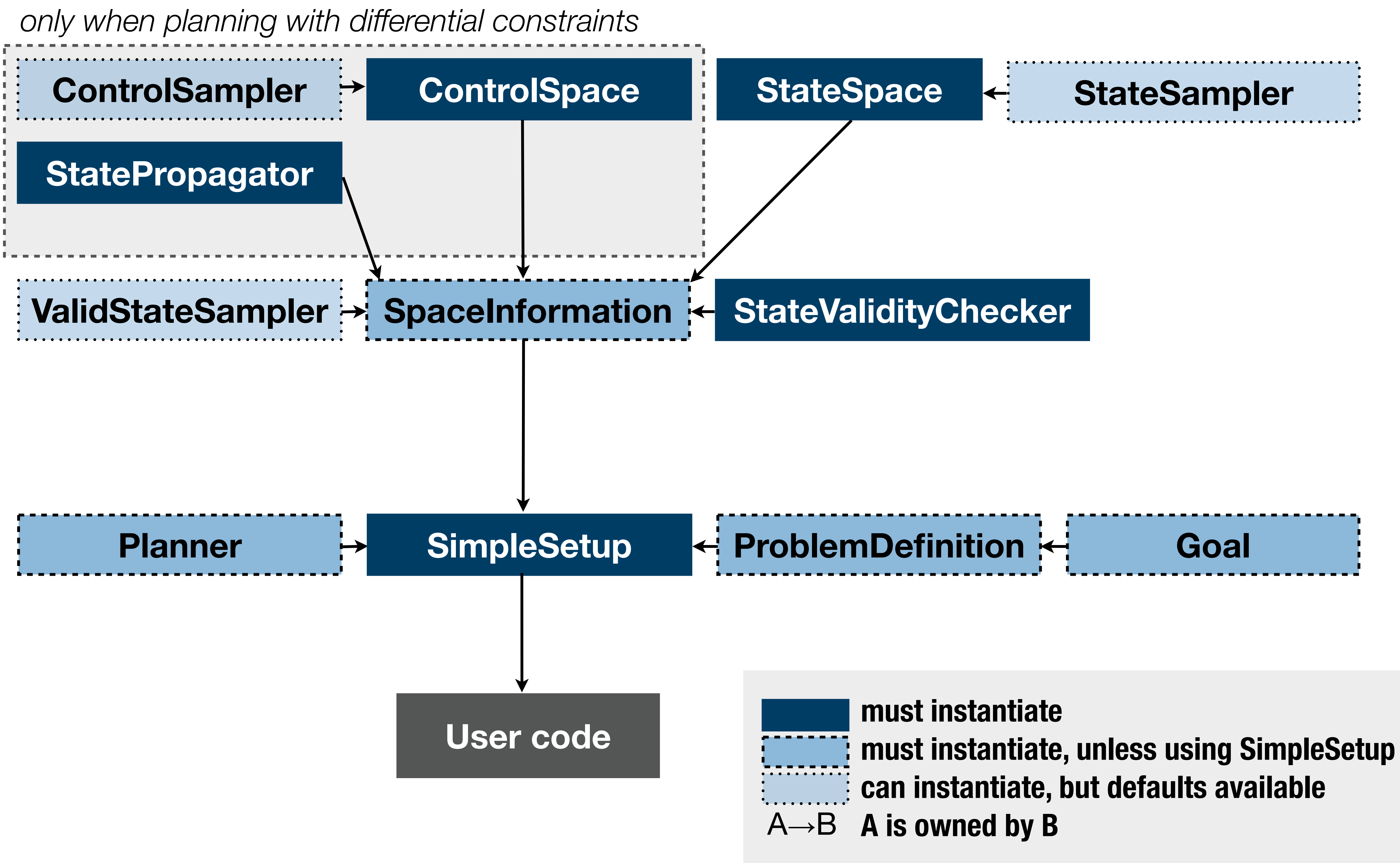
Many planners available in OMPL



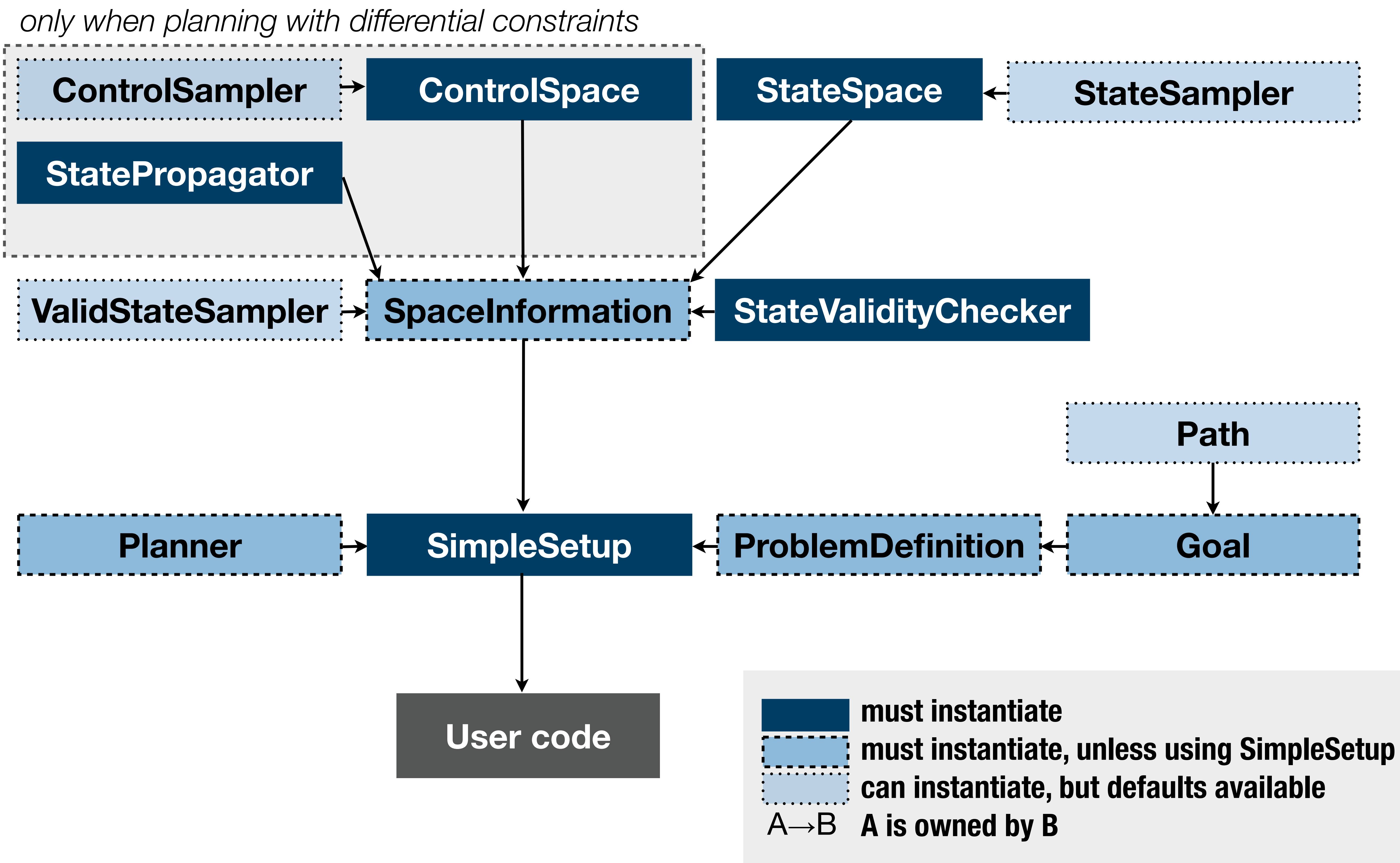
API overview



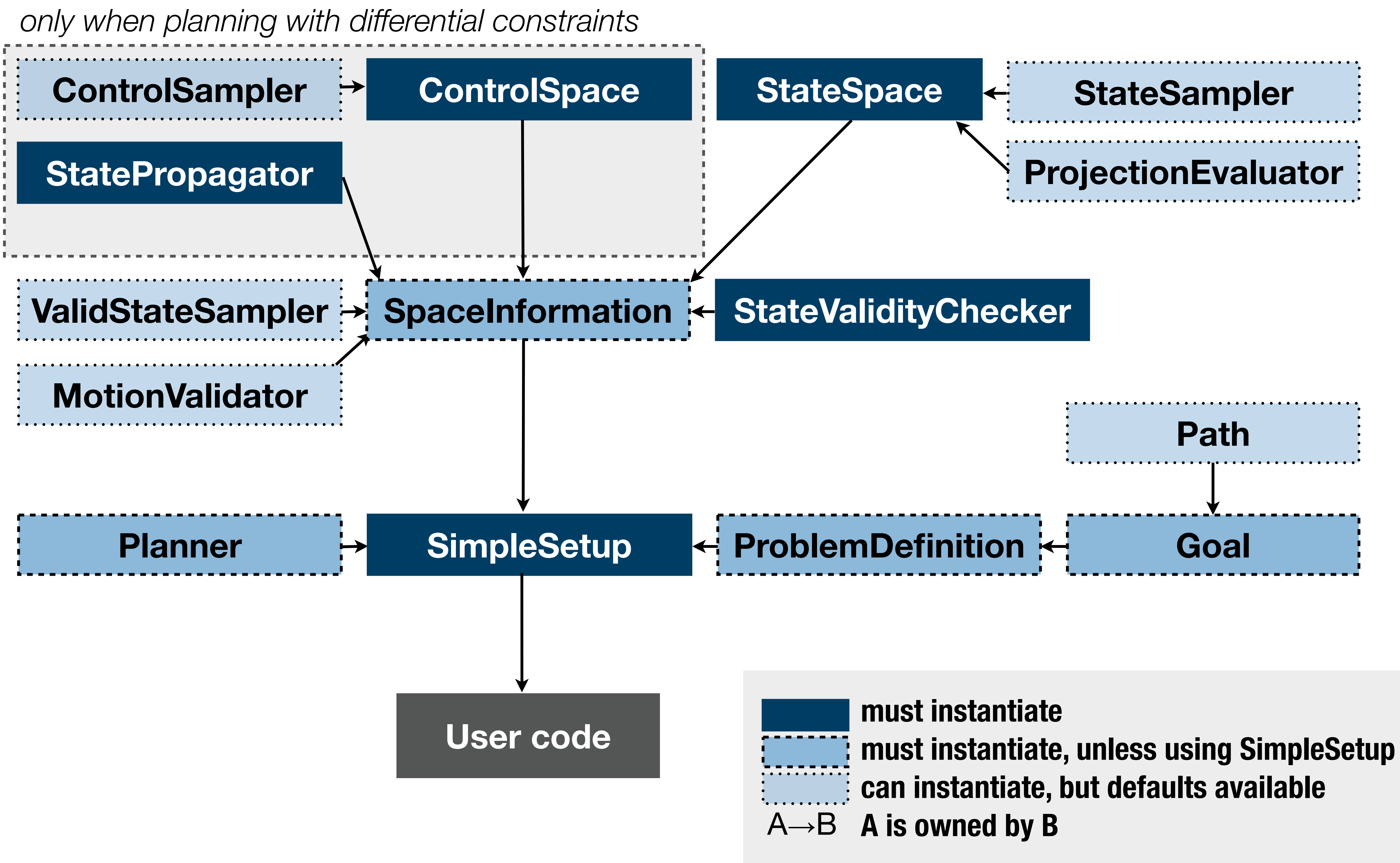
API overview



API overview



API overview



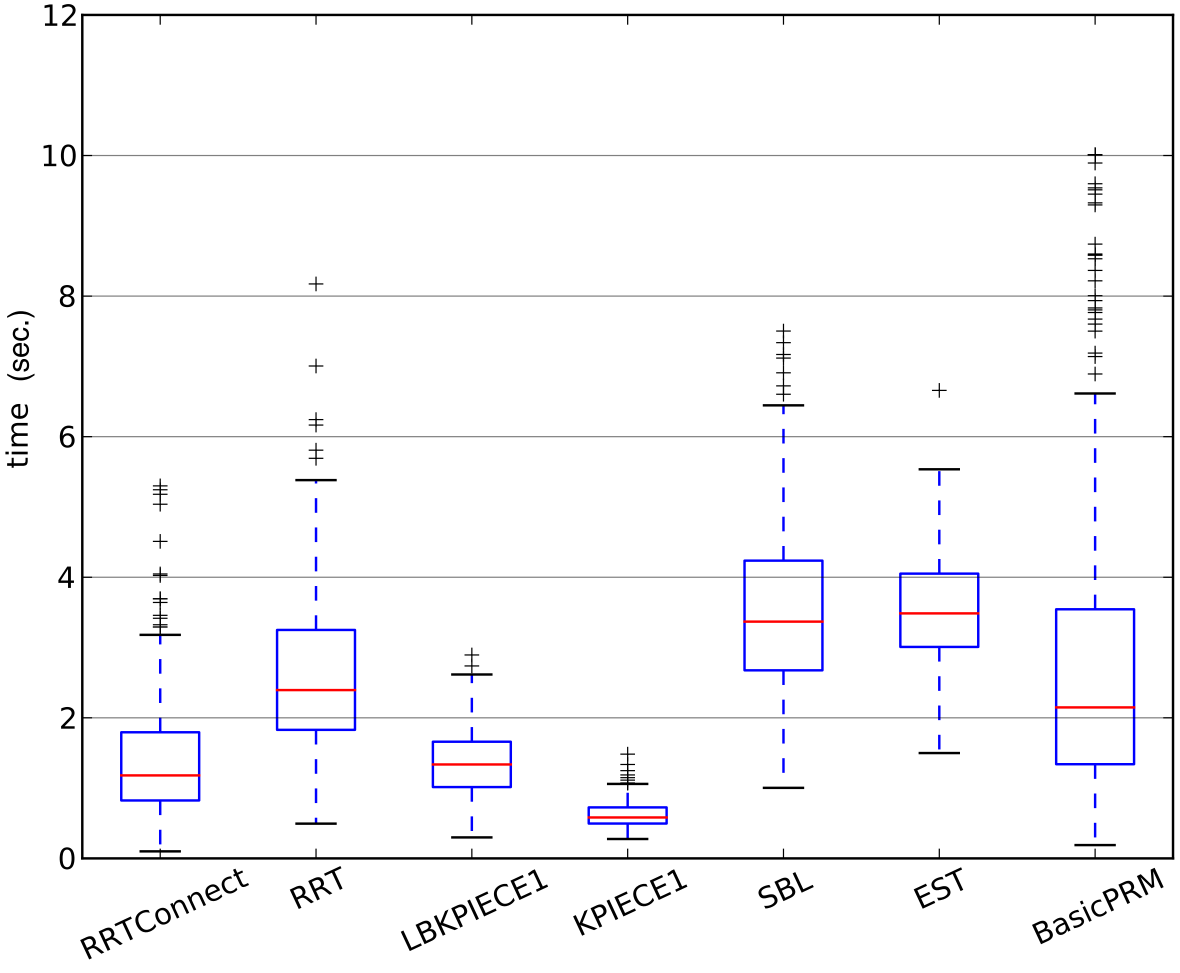
Minimal code example

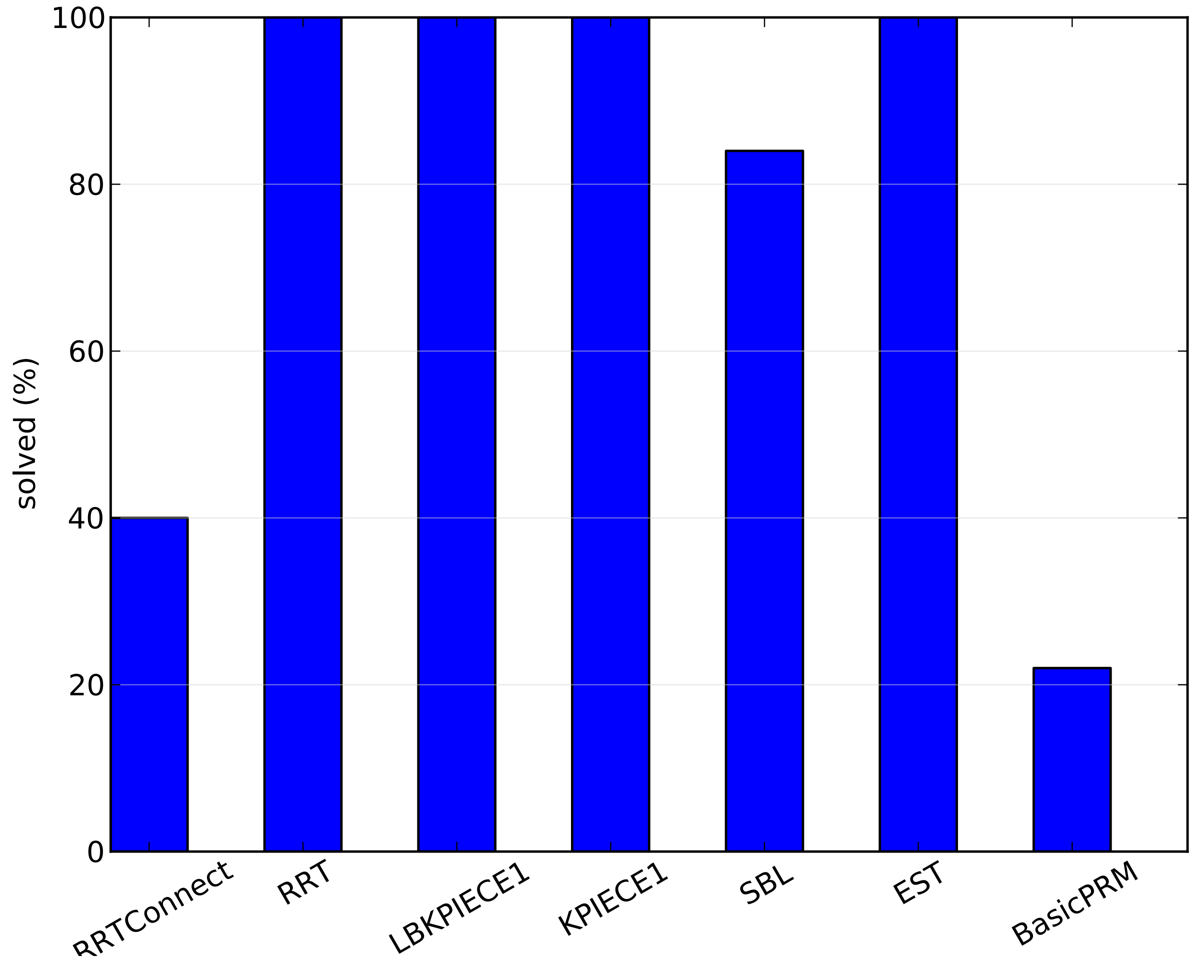
```
space = SE3StateSpace()
# set the bounds (code omitted)

ss = SimpleSetup(space)
# "isStateValid" is a user-supplied function
ss.setStateValidityChecker(isStateValid)

start = State(space)
goal = State(space)
# set the start & goal states to some values
# (code omitted)

ss.setStartAndGoalStates(start, goal)
solved = ss.solve(1.0)
if solved:
    print setup.getSolutionPath()
```

Minimal code example

```
StateSpacePtr space(new SE3StateSpace());
// set the bounds (code omitted)

SimpleSetup ss(space);
// "isStateValid" is a user-supplied function
ss.setStateValidityChecker(isStateValid);

ScopedState<SE3StateSpace> start(space);
ScopedState<SE3StateSpace> goal(space);
// set the start & goal states to some values
// (code omitted)

ss.setStartAndGoalStates(start, goal);
bool solved = ss.solve(1.0);
if (solved)
    setup.getSolutionPath().print(std::cout);
```

Benchmarking





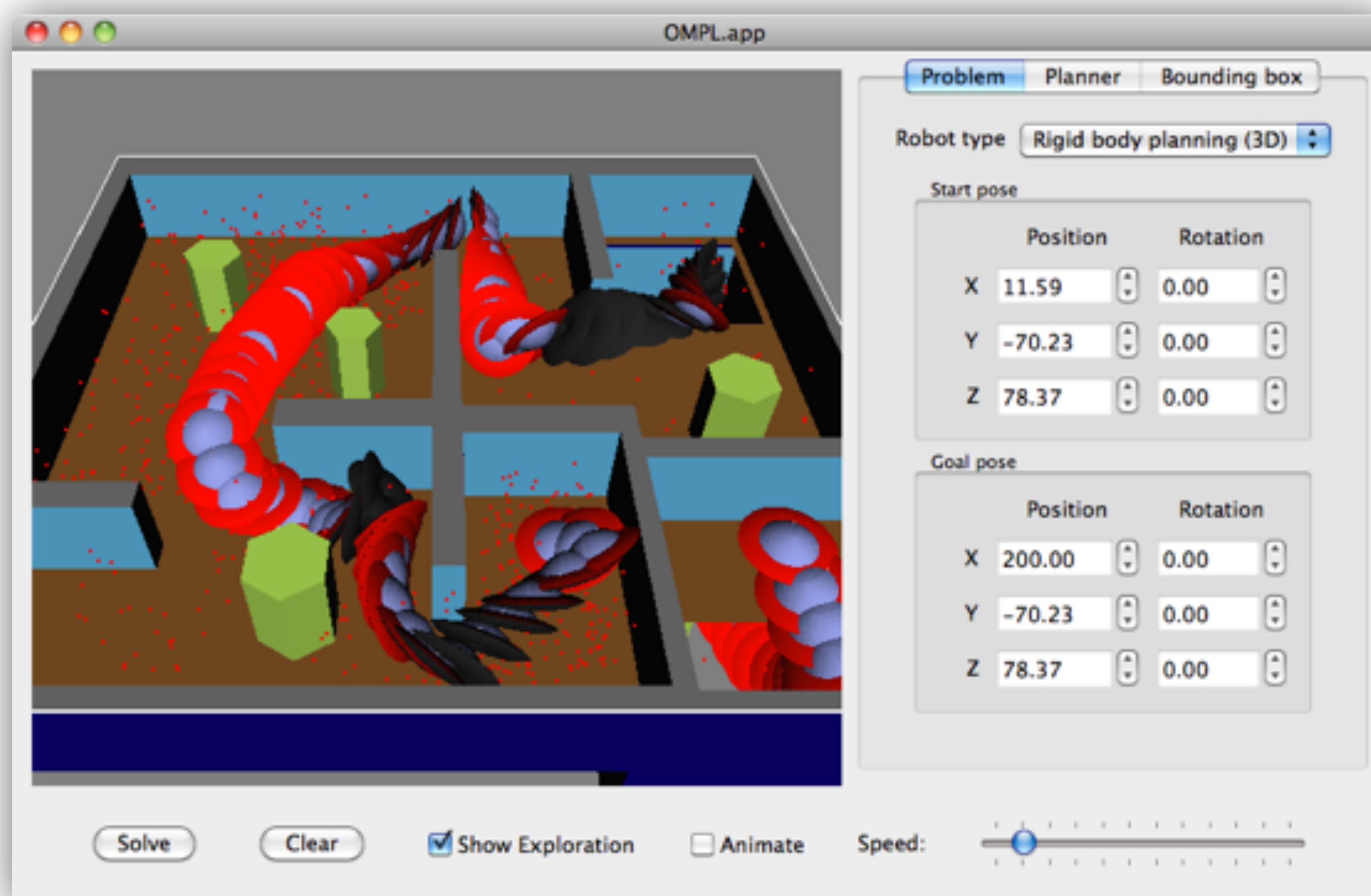
Benchmarking

```
SimpleSetup setup;
// motion planning problem setup code omitted
Benchmark b(setup, “My First Benchmark”);

b.addPlanner(base::PlannerPtr(new geometric::RRT(setup.getSpaceInformation())));
b.addPlanner(base::PlannerPtr(new geometric::KPIECE1(setup.getSpaceInformation())));
b.addPlanner(base::PlannerPtr(new geometric::SBL(setup.getSpaceInformation())));
b.addPlanner(base::PlannerPtr(new geometric::EST(setup.getSpaceInformation())));
b.addPlanner(base::PlannerPtr(new geometric::PRM(setup.getSpaceInformation())));

b.benchmark(runtime_limit, memory_limit, run_count, true);
b.saveResultsToFile();
```

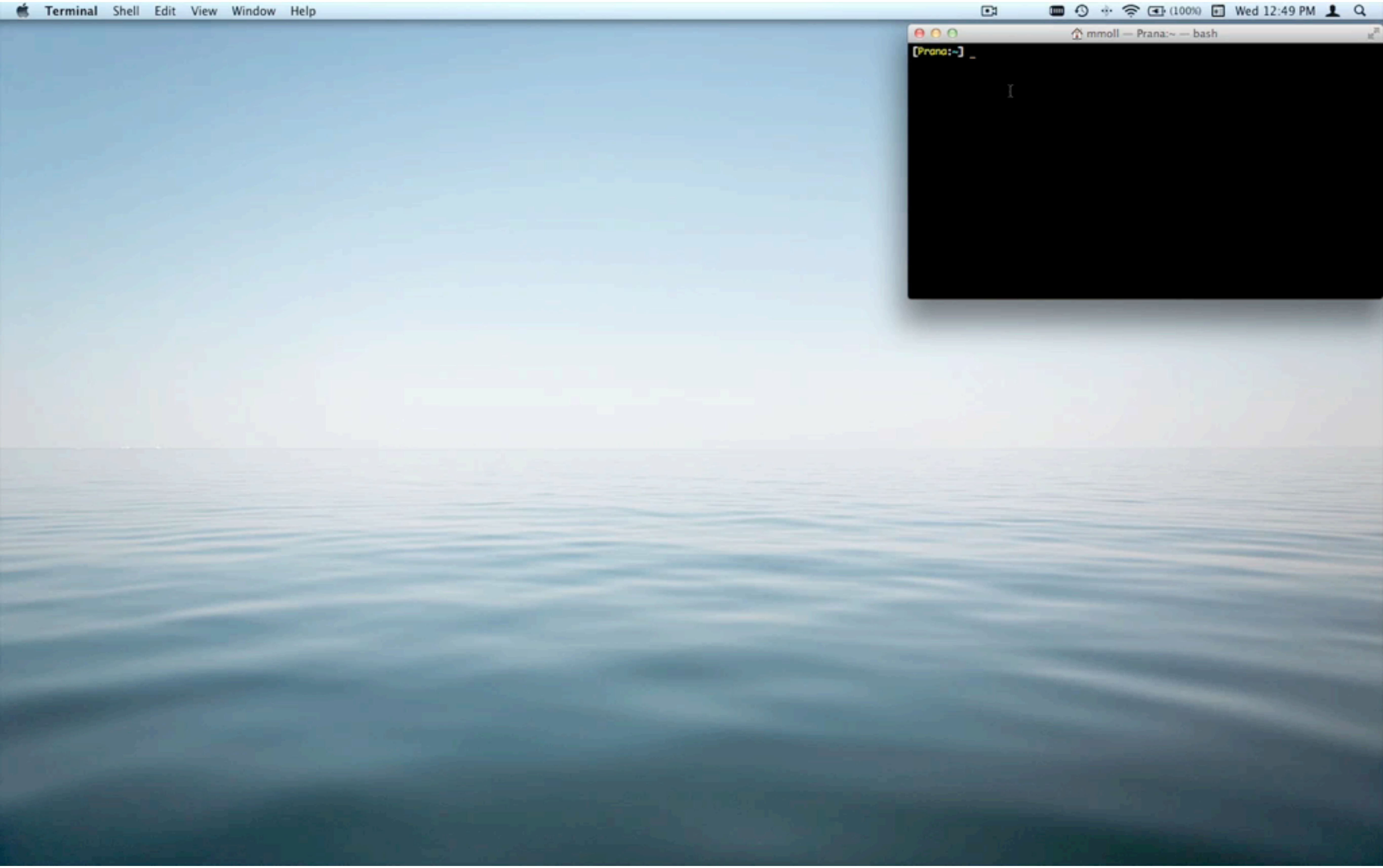
Script post-processes benchmark log files to create/update SQLite database and plots

OMPL.app

- Front-end that demonstrates integration with libraries for collision checking, 3D mesh loading, GUI toolkit
- Easy-to-use tool for novices to get started
- Alternative to ompl_ros_interface

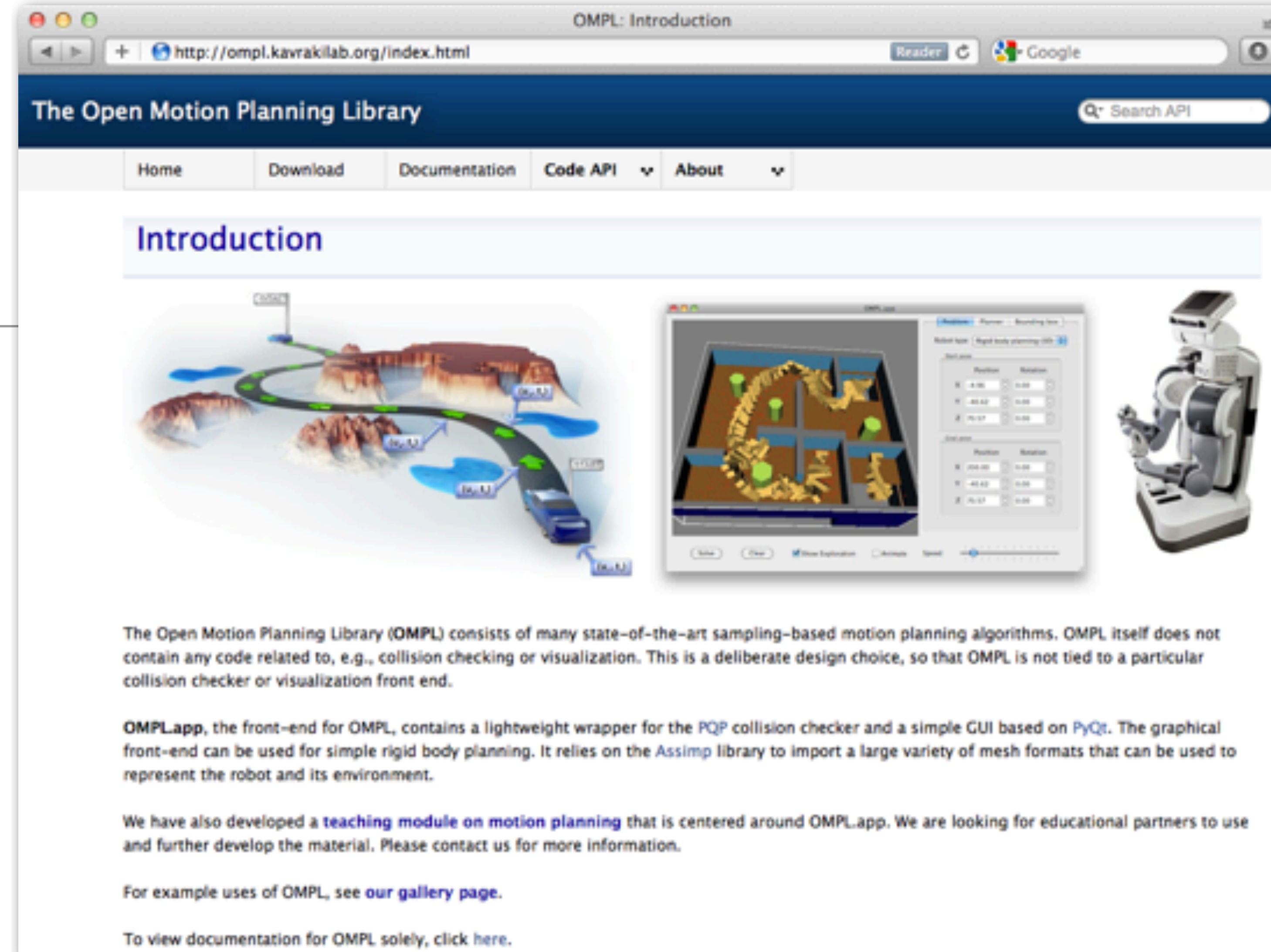


OMPL.app demo / screencast


Terminal
Shell
Edit
View
Window
Help
(100%)
Wed 12:49 PM
mmoll — Prana:~ — bash
[Prana:~]

Resources to get started with OMPL

OMPL online



- **Web site:** http://ompl.kavrakilab.org

- **Mailing lists:**
  - Developers: ompl-devel@lists.sourceforge.net
  - Users: ompl-users@lists.sourceforge.net

- **Public Mercurial repository:** http://ompl.hg.sourceforge.net:8000/hgroot/ompl/ompl

OMPL for education

- Programming assignments centered around OMPL, available upon request.
- Ongoing educational assessment.
- Already in use in several robotics / motion planning classes.

Happy OMPL users: students in the Algorithmic Robotics class at Rice, Fall 2010

OMPL tutorials

Step-by-step walkthroughs for:

- geometric planning for rigid body in 3D
- working with states and state spaces
- representing goals
- benchmarking
- creating new planning algorithms

OMPL examples

- Many demos for basic usage patterns, often available in both C++ and Python

- Demos for advanced features:

  1. **Lazy goal sampler, generic numerical IK solver**

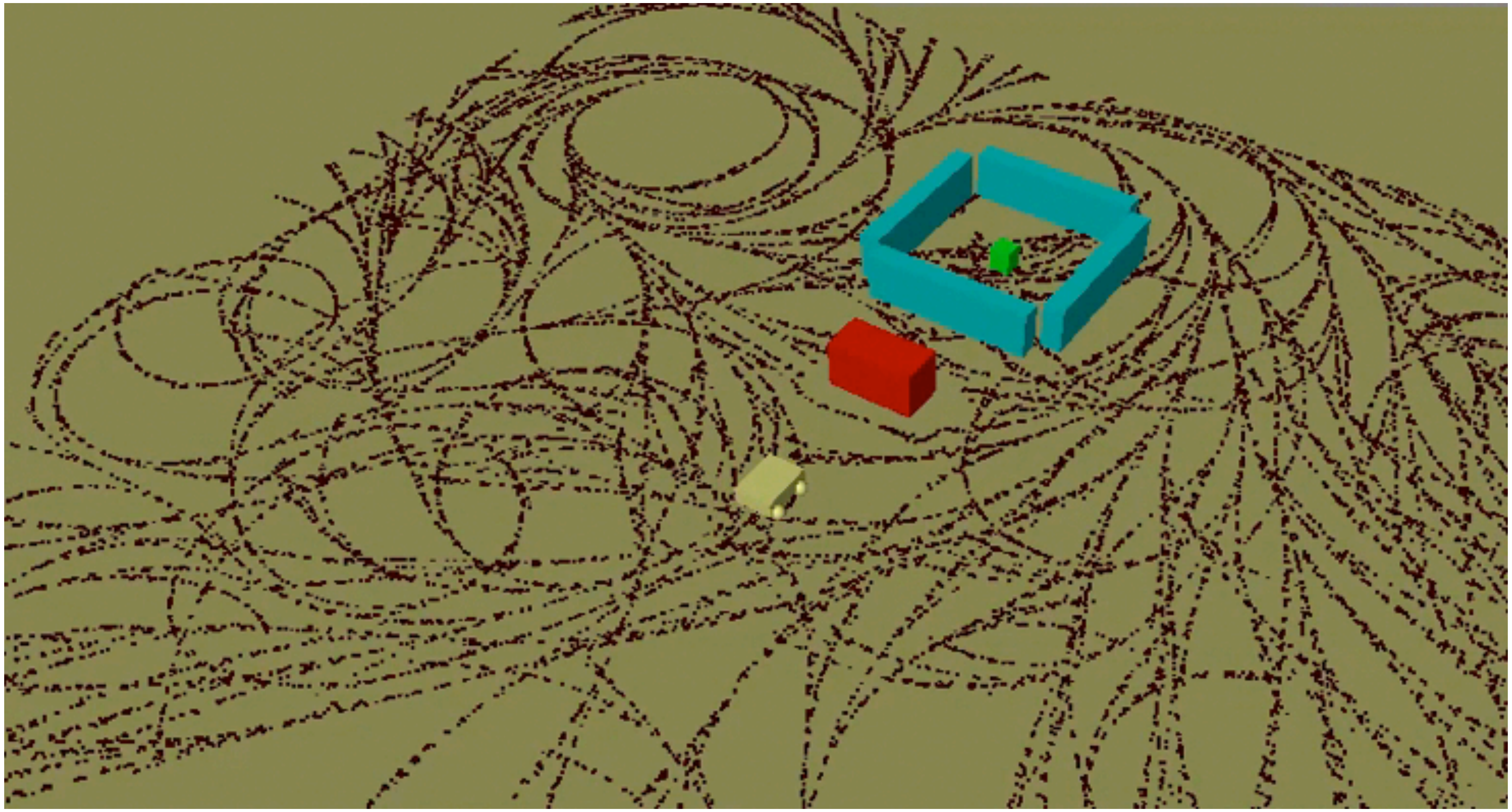
  2. **Using the Open Dynamics Engine**

Example 1: lazy goal sampler + IK

- Spawn thread responsible for generating goal states
  - generate as many goal states as user wants
  - OMPL comes with Genetic Algorithm-based IK solver, but other types of solvers can be used
- Planner waits until at least one goal state is available
- Can use bi-directional planner with implicit goal region in state space
- Same approach is used in ROS for end-effector constraints

Example 2: OMPL + ODE

- Treat ODE physics engine as a black box state propagation function: Given state, controls, and time duration, ODE produces new state
- Can plan for systems with movable objects, various contact modes, etc.
- Same approach can be used for other physics engines



Discussion

- OMPL actively developed, but ready for general use
- Can easily implement new algorithms from many reusable components
- Simple high-level interface:
  - Can treat motion planner almost as a black box
  - Easy enough that non-experts can use it
- Interface generic enough to be extensible in many ways

We want your contributions!

Acknowledgements

Rice University:

Ioan Şucan

Lydia Kavraki

Matt Maly

Devin Grady

Bryant Gipson

Amit Bhatia

Willow Garage:

Sachin Chitta

Gil Jones

Funding from:

NSF CCLI grant #0920721

Willow Garage