ROS-Industrial Advanced Developer's Training Class

July 2023

Southwest Research Institute

Advanced Topic:
Motion Planning with Tesseract

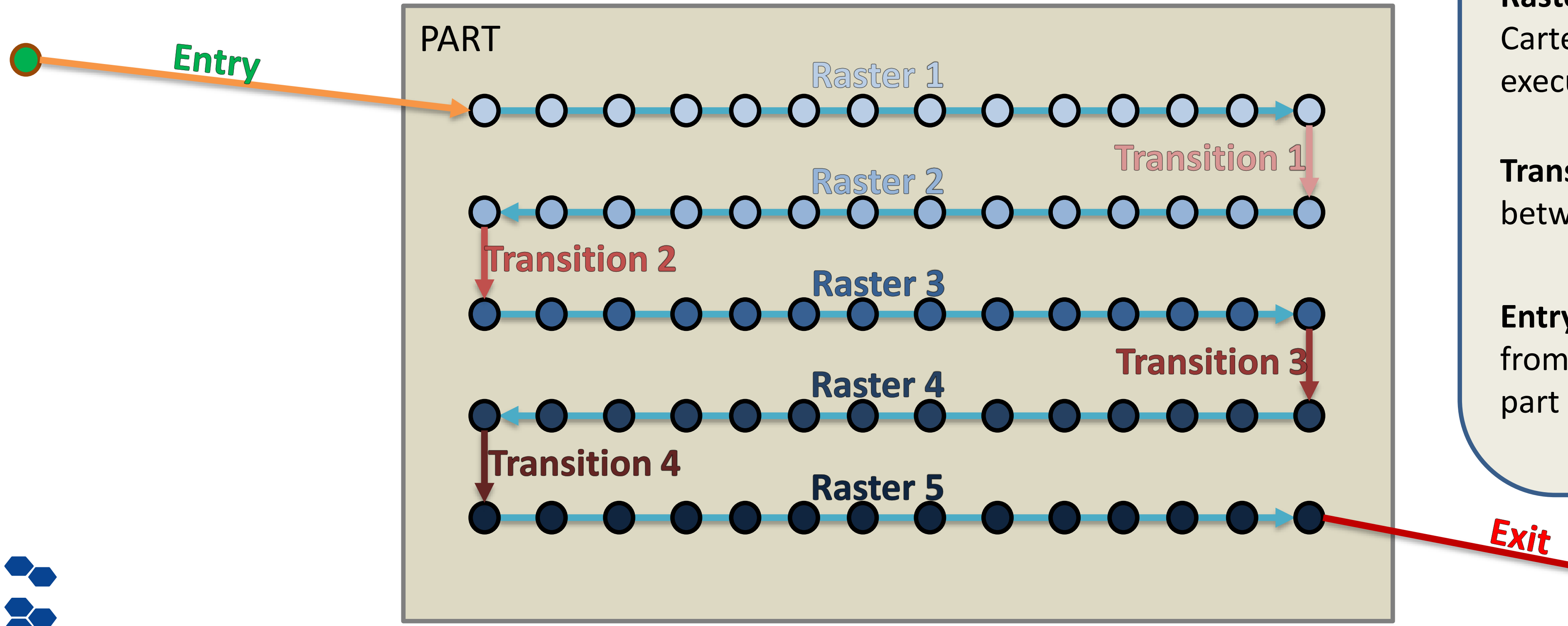
Southwest Research Institute

What is Tesseract?

- A ROS independent robotic manipulator environment
- Runs motion planning and collision checking efficiently
- Dynamic scene graph
  - Add, remove, or move links anywhere in the environment scene
- Highly customizable parallel planning
  - Create and customize pipelines
  - Create and customize individual tasks
- https://github.com/tesseract-robotics/tesseract

Motion Planning Goal

Generate a robot trajectory to execute a toolpath



Definitions

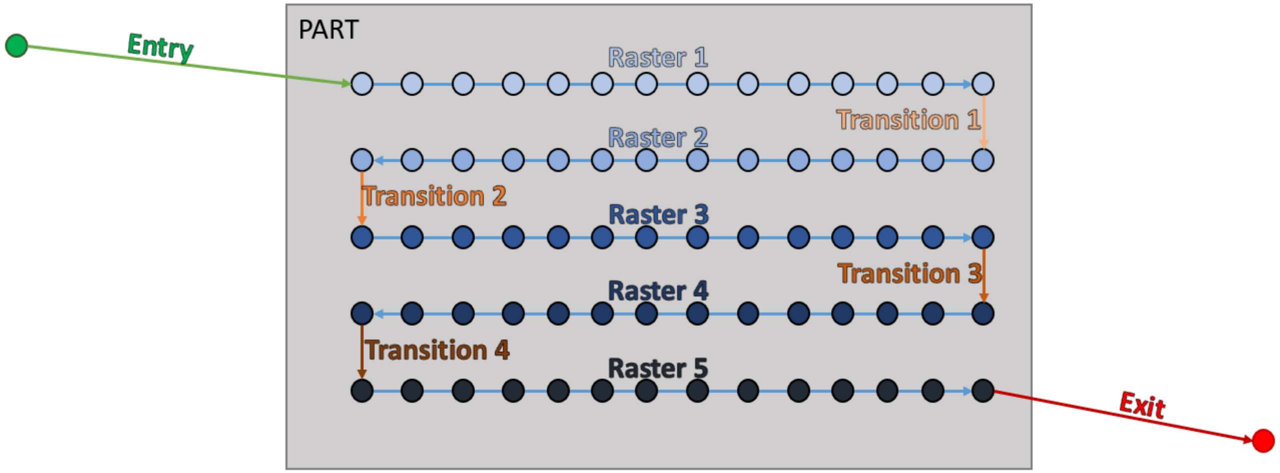
Trajectory - A series of joint states (position, velocity, acceleration, and time stamp) strung together

Toolpath - A collection of one or more rasters

Raster - A series of specified Cartesian waypoints to be executed without breaking*

Transition - A freespace move between rasters

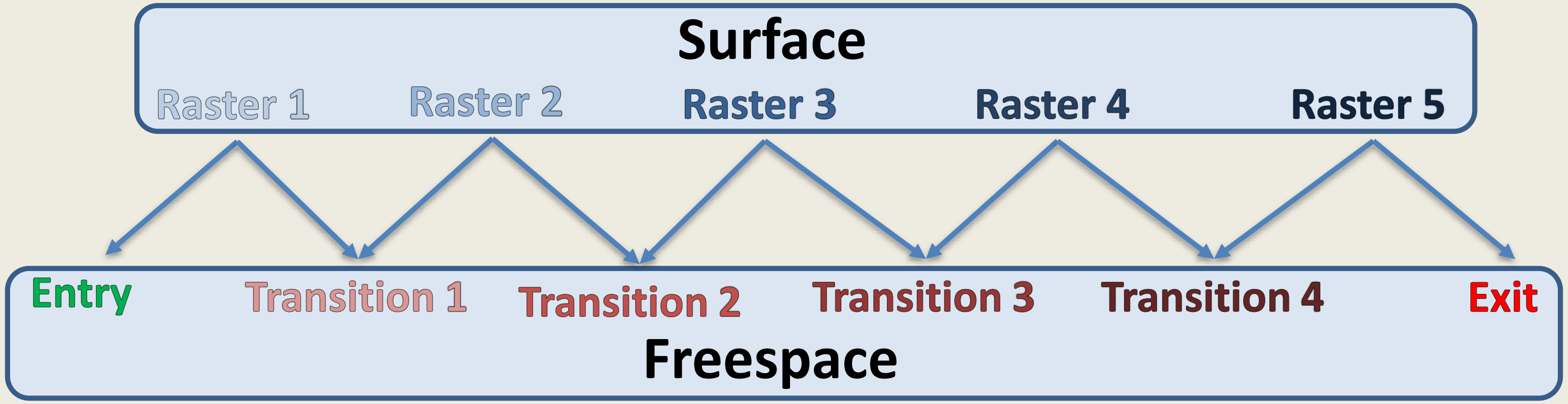
Entry/Exit - A freespace move from/to a position away from the part



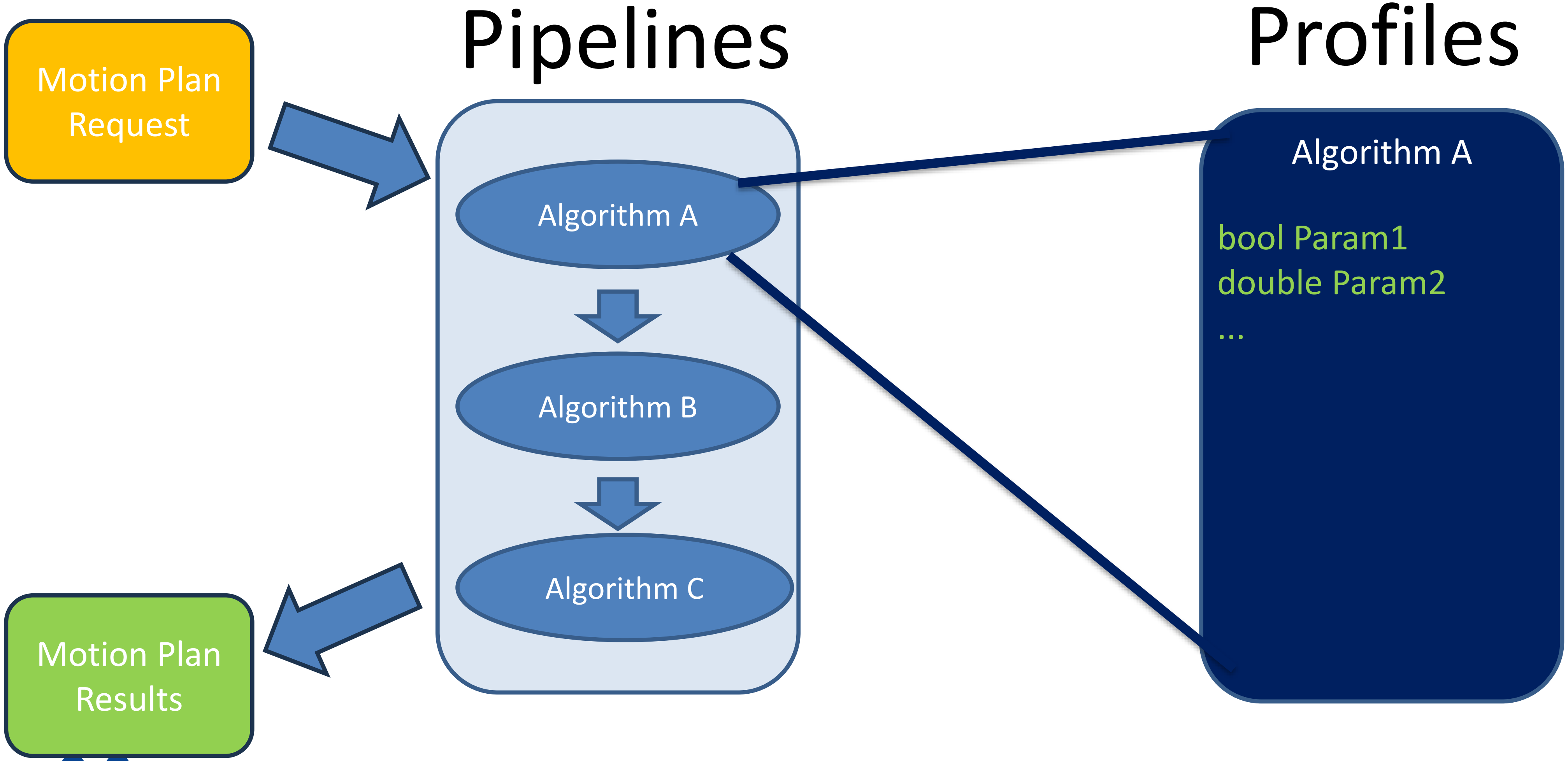
Execution Order

Entry
Raster 1
Transition 1
Raster 2
Transition 2
Raster 3
Transition 3
Raster 4
Transition 4
Raster 5
Exit

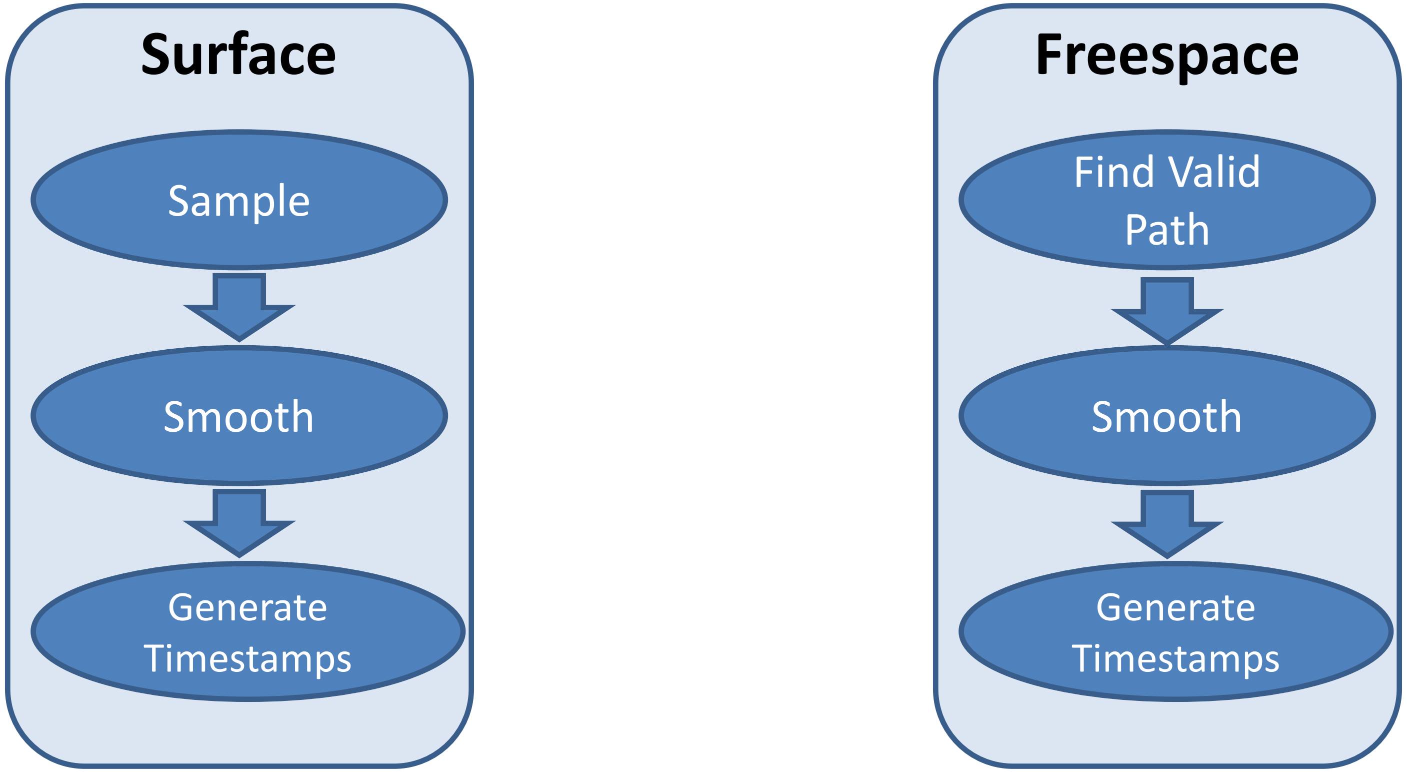
Planning Dependency Graph



Two main components to solving motion plans in Tesseract



Workflows



Surface

Sample

Smooth

Generate Timestamps

Freespace

Find Valid Path

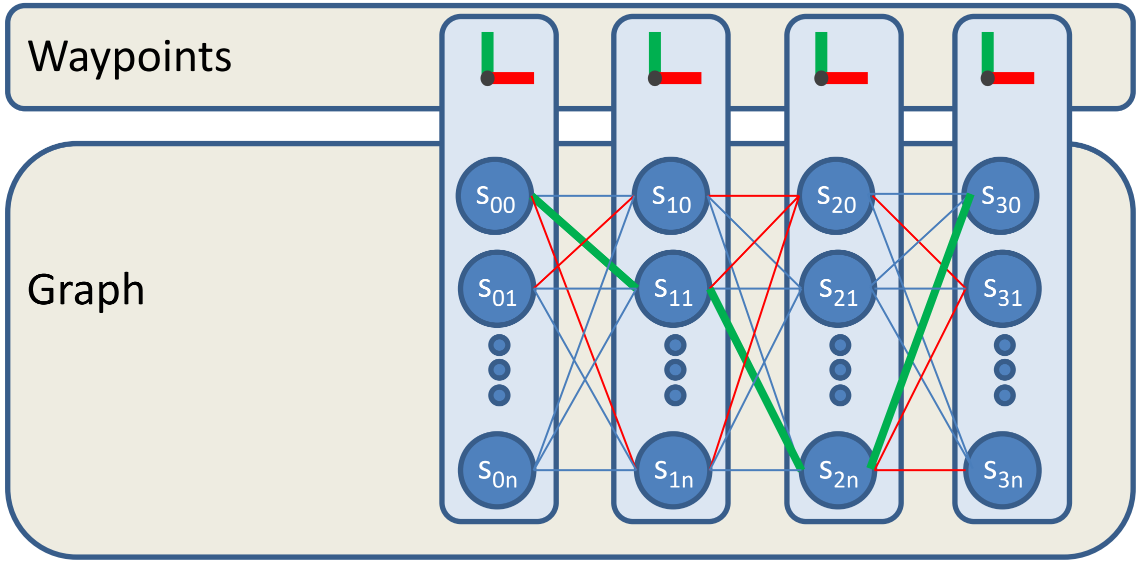
Smooth

Generate Timestamps

Surface Planning - Sampling

Descartes (https://github.com/swri-robotics/descartes_light)

- **Input**: A series of Cartesian waypoints
- **Output**: Series of joint positions



s_{xy}: a valid robot position for a given waypoint

╲ : a valid connection

╲ : an invalid connection

╲ : optimal path

Descartes Parameters

- Inverse kinematics solver (waypoint -> joint state)
- Waypoint sampler
  - Fixed -> n solutions per waypoint, generally 8
  - Axial -> (n)*(360°/ sample angle)
  - Extra axis (7 DOF/rail/gantry)-> n*(extra axis$_1$ samples)*(extra axis$_2$ samples)*…
- Vertex evaluator
  - Account for certain configurations that may be in violation (DCS on Fanuc)
- Edge evaluator
  - Account for joint flips
- Environment collision checker
  - Specify allowed collision distance

Surface Planning - Smoothing

TrajOpt (https://github.com/tesseract-robotics/trajopt)

- **Input:** Seed trajectory
- **Output:** Trajectory that is smooth, collision-free, or meets other specified criteria
- **Functionality:**
  - Works by leveraging optimization algorithms
  - Use costs and constraints

TrajOpt Parameters

All parameters have a coefficient that can be increased/decreased to change it's influence

- Collision parameters (cost or constraint)
  - Use weighted sum – combines collisions to be a single term
  - Safety margin – how far of collision distance must be maintained
  - Safety margin buffer – distance beyond safety margin to still use in optimization
- Smoothing (cost)
  - Velocity
  - Acceleration
  - Jerk
- Joint/Cartesian (cost or constraint) – Set a specified joint or cartesian DOF to be more or less valued
  - Example: Set the 6th term in the Cartesian coefficient to be 0 to allow rotation about the z axis
- Avoid singularities (cost)
- Longest valid segment – Resolution to check validity of state (as opposed to just checking discretely at each point in the seed)
- Other user defined parameters (cost or constraint)

Surface Planning – Generate Timestamps

Iterative Spline Parameterization

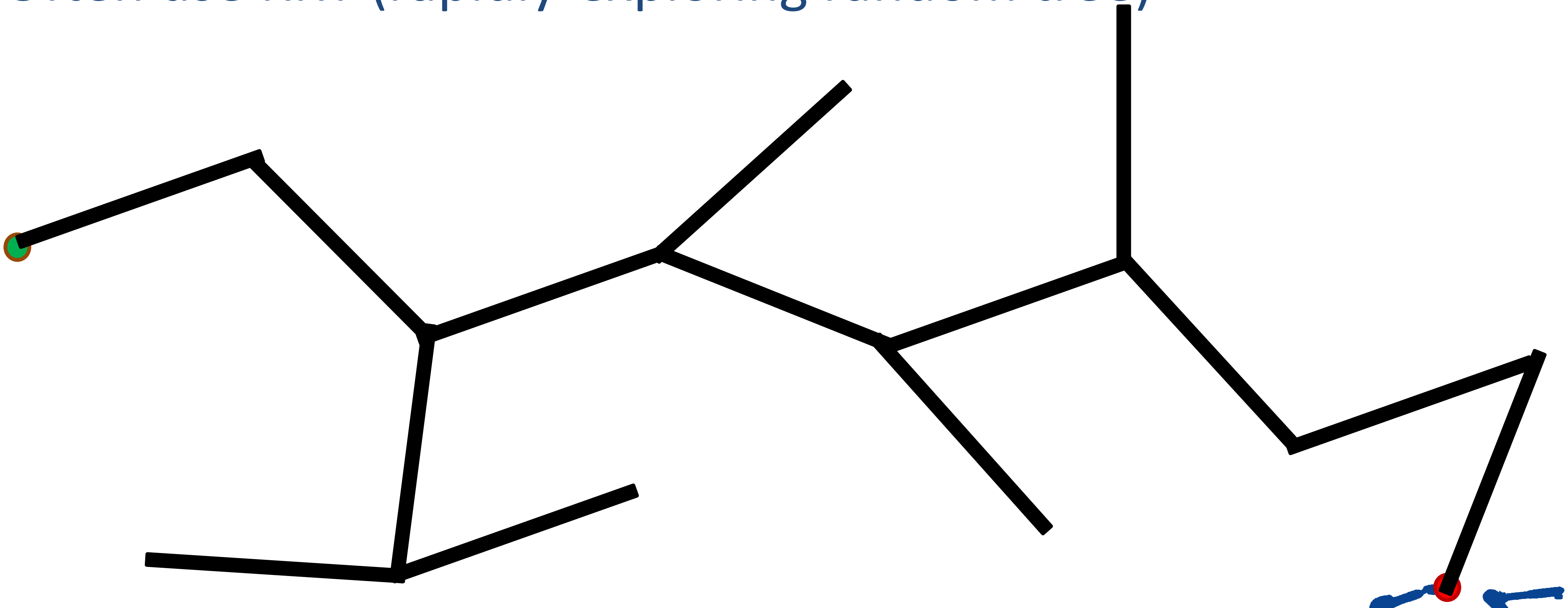
- **Input:** Seed trajectory
- **Output:** Trajectory with timestamps that will not violate any robot constraints
- **Parameters:**
  - Joint max velocities
  - Joint max accelerations

Freespace Planning – Find Valid Path

- **Input:** Start and end state
- **Output:** Valid trajectory between the states
- **Methods:**
  1. Joint interpolated motion
     - Good for very short and simple motions
  2. TrajOpt
     - Good for slightly more complex motions that would otherwise be in collision
  3. OMPL (https://ompl.kavrakilab.org/)
     - Good for navigating when complex motions are needed

OMPL

- Many planning algorithms
- Often use RRT (rapidly-exploring random tree)



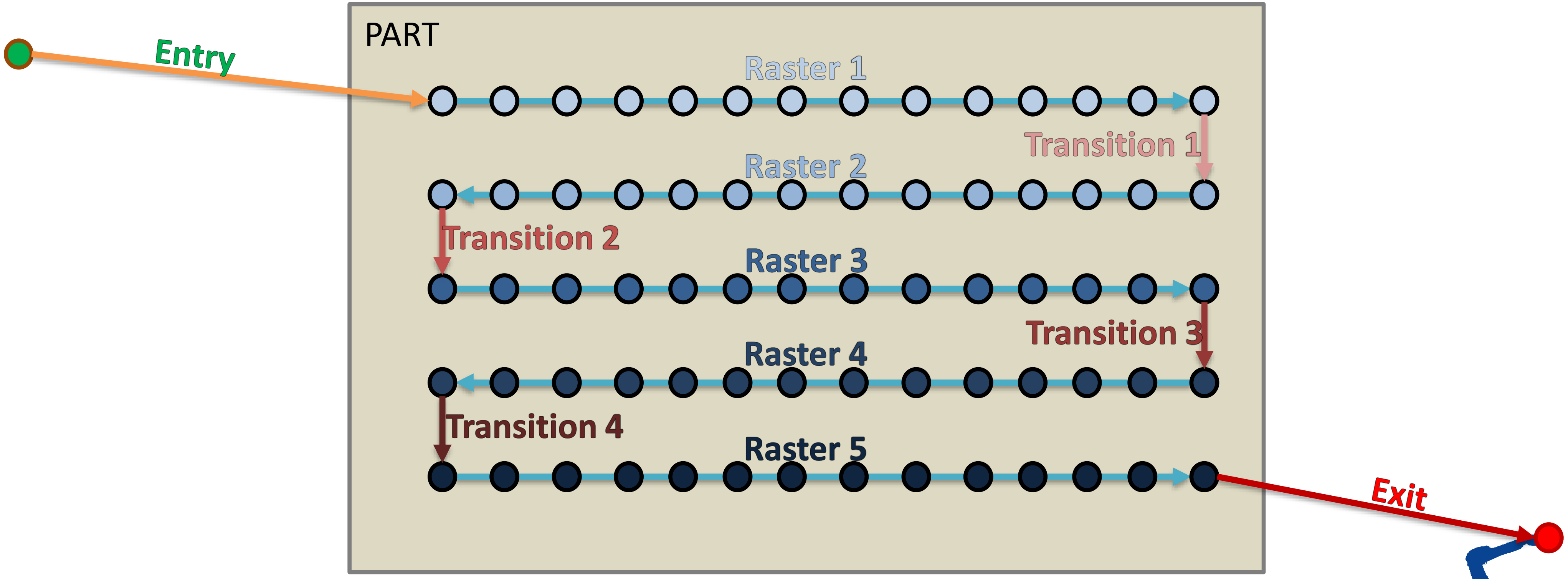
OMPL Planner Types

- RRT
  - As seen on previous slide
  - Parameters
    - Range: how long each step size is
      - Longer range solves big open spaces faster
      - Smaller range helps get through tight spots
    - Goal bias: How frequently the algorithm tries to move to the goal
- RRT-Connect (most commonly used by SwRI)
  - Build a tree from each side and try to *Connect* them
  - Parameters
    - Range (same as above)
- See more at https://ompl.kavrakilab.org/planners.html

Freespace Planning – Smoothing & Timestamps

Same as surface planning

Questions?


PART
Entry
Raster 1
Transition 1
Raster 2
Transition 2
Raster 3
Transition 3
Raster 4
Transition 4
Raster 5
Exit