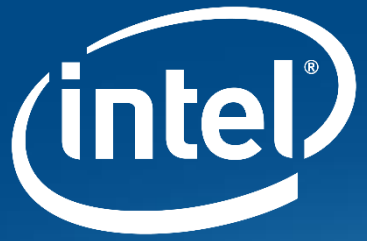


Navigation2 Overview

Matt Hansen

Sr. Robotics SW Architect

Intel Open Source Robotics

Intel Open Source Robotics

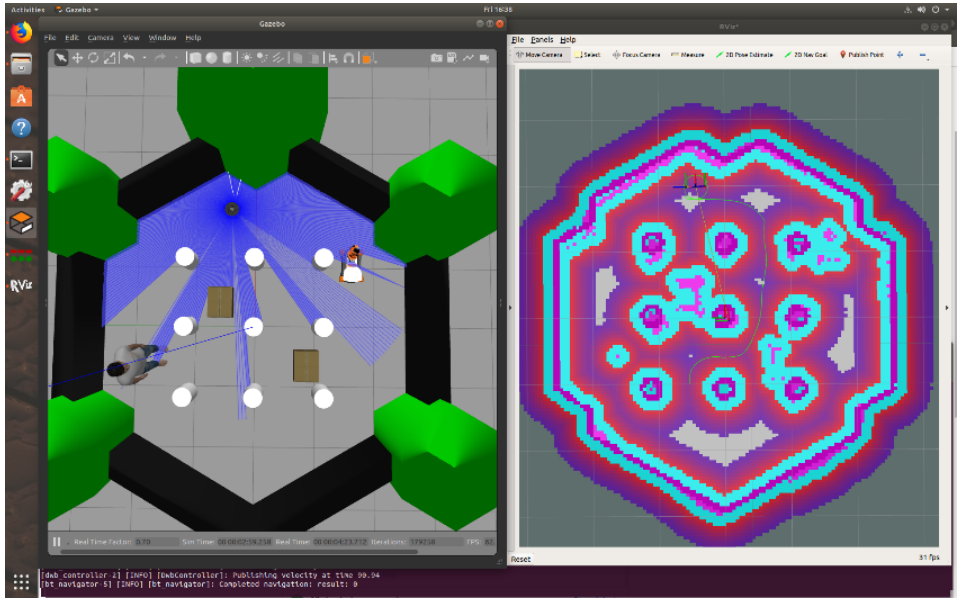
Strategy:
Enable Intel technology pillars in Robotics applications through ROS2

Make it easy for you to INNOVATE

Navigation2 Overview

ROS Navigation -http://wiki.ros.org/navigation

- One of the key and most used packages of ROS
- Autonomous movement for a robot in a 2D map
  - Given a ‘current pose’ and a ‘goal ‘pose’
  - Path is planned, robot drives itself to the goal



- Key to accelerating ROS2 development and adoption across the community and industry
  - No one had committed to porting Navigation to ROS2 as of Spring 2018
  - Proactively, our team assumed ownership of ROS2 Navigation in 2018
  - Ported, refactored, and made architectural improvements from ROS
- https://github.com/ros-planning/navigation2

Navigation2 Requirements

In 2018, we created a ROS Discourse topic to gather input from the ROS community; what changes and improvements they would like to see in ROS2 Navigation.

Some recurring themes emerged:

- Customizable logic –ability to customize behavior, less need to fork the code
- Modularity –ability to more easily replace planners and control algorithms
- Extensibility – ability to use Python or other languages to write planners and control

In addition, the development team wanted to ensure other properties such as:

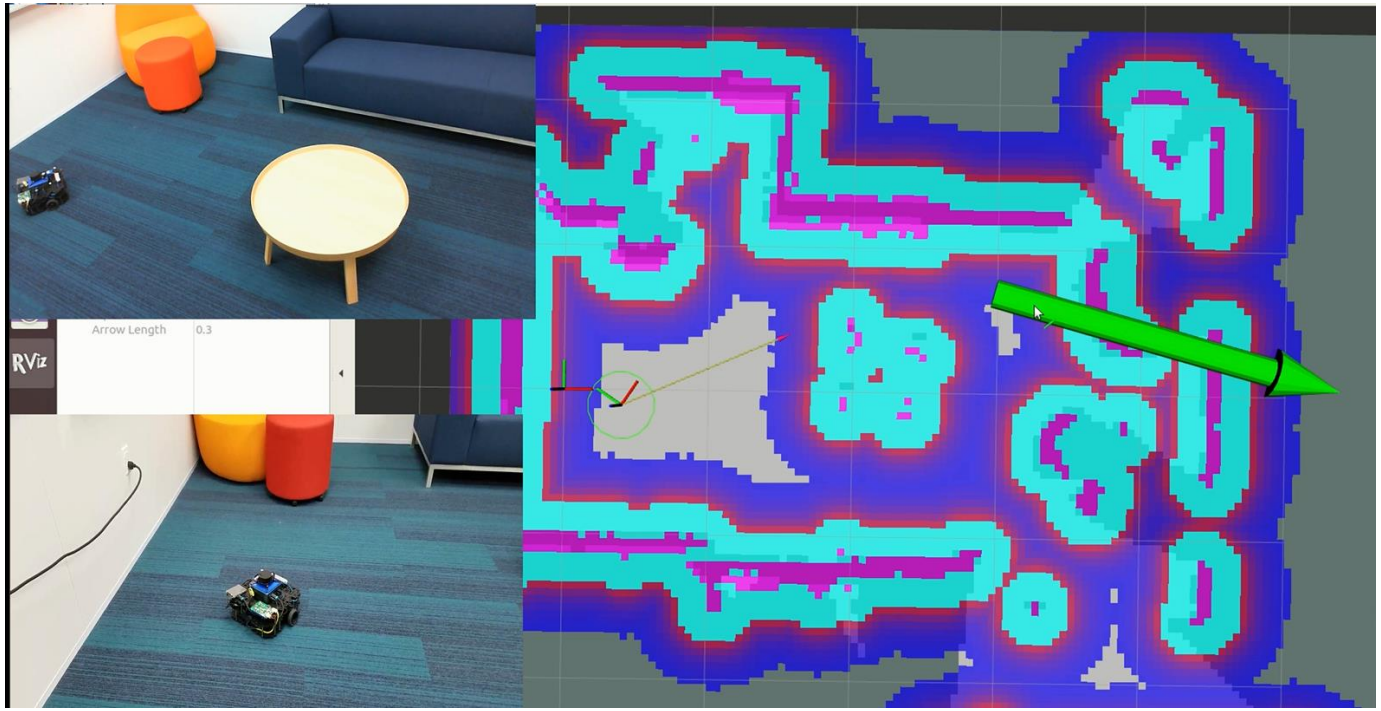
- Reliability – the system should be able to perform in a consistent way
- Quality – the code submitted should be validated before merging
- Maintainability – the workflow should prevent regressions in the above

The navigation2 project is an attempt to meet these goals

Navigation2 Architecture Improvements

Extensibility
Flexibility
Modularization
Reliability

- Behavior Trees
- Planners as ROS2 Actions
- Recovery Behaviors as ROS2 Actions
- Lifecycle nodes for systematic launch



[Demo Video: ROS2 Navigation]

Comparison – ROS Navigation vs Navigation2

amcl and map_server – **ported** from ROS Navigation with refactoring

move_base – **replaced by behavior tree** based navigation node called ‘bt_navigator’

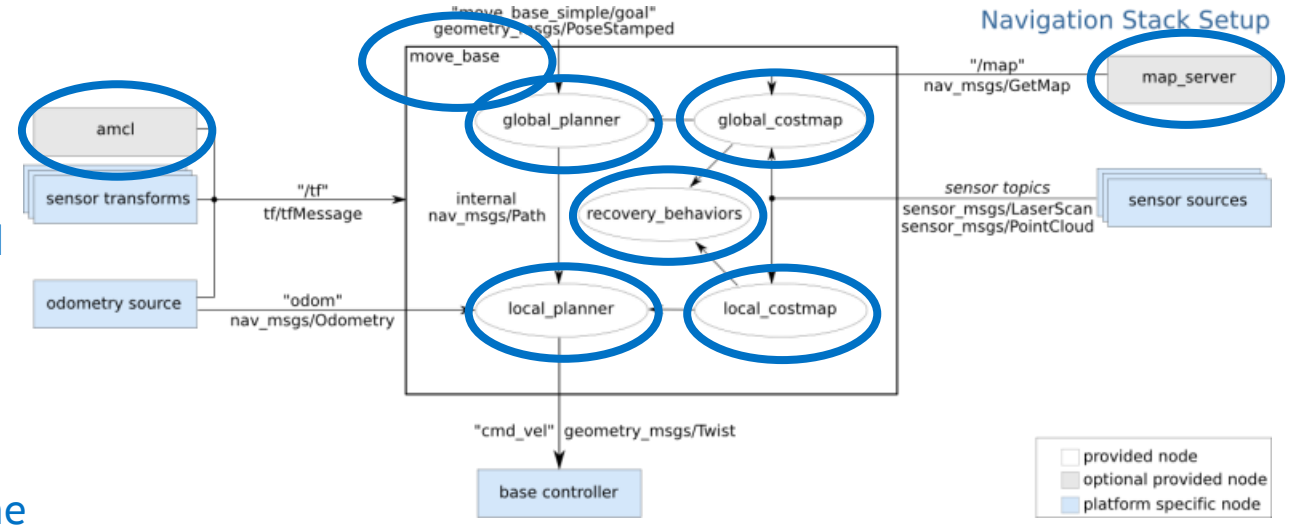
recovery_behaviors – now **actions** within the behavior tree(s)

global_planner – **navfn ported** as a global planner called navfn_planner

local_planner – ‘dwb’ local planner **ported** from the robot_navigation project as dwb_planner

global_costmap and local_costmap - contained within the global and local planners respectively

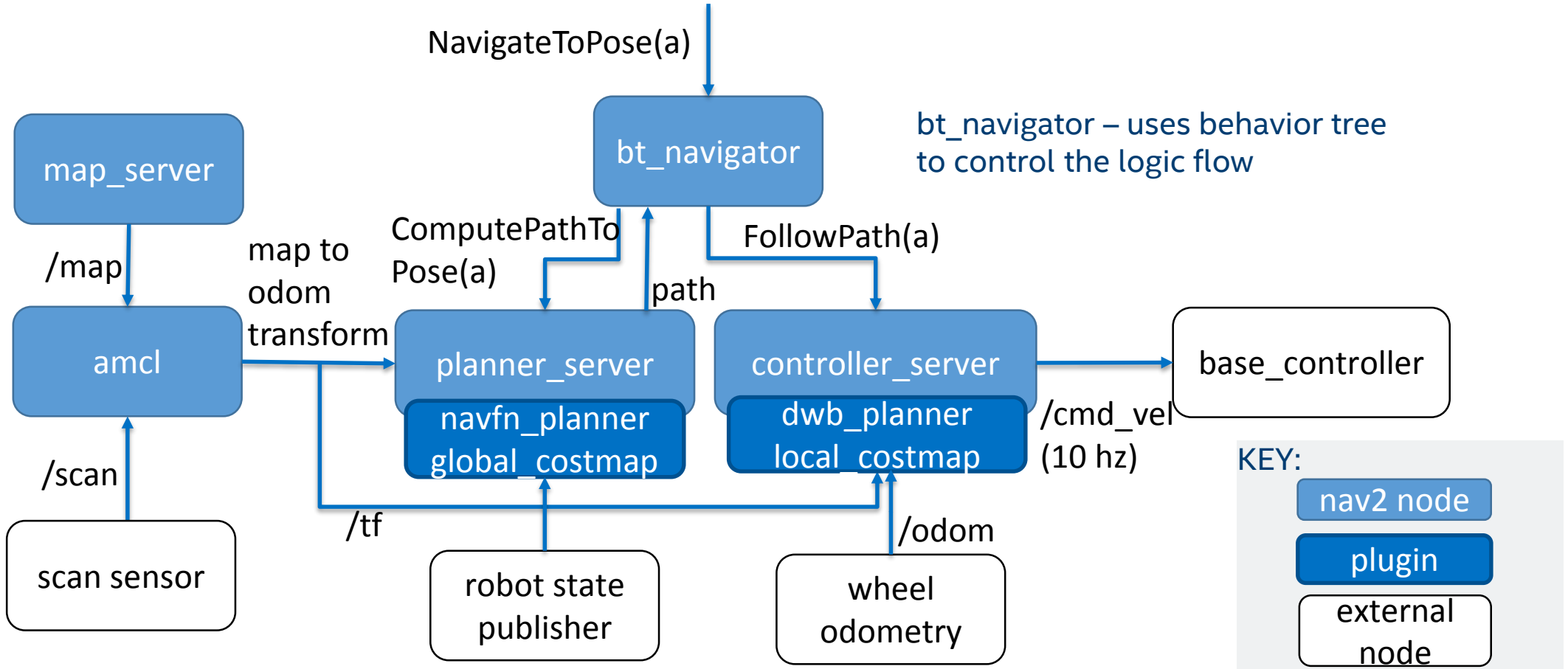
planner_server and controller_server **(NEW)** - ROS2 action servers (ComputePathToPose) and (FollowPath)



http://wiki.ros.org/navigation/Tutorials/RobotSetup

We blew up move_base and planted a behavior tree in it’s place

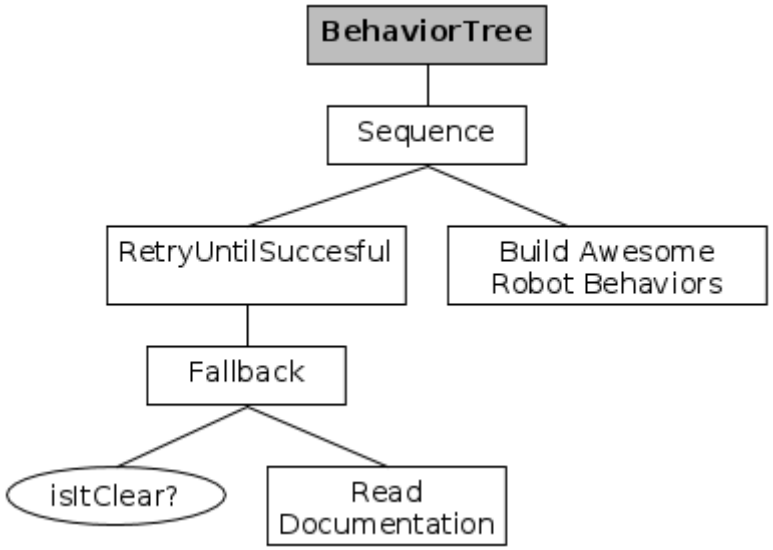
Navigation2 Architecture Overview ROS API



Behavior Trees

What are behavior trees?

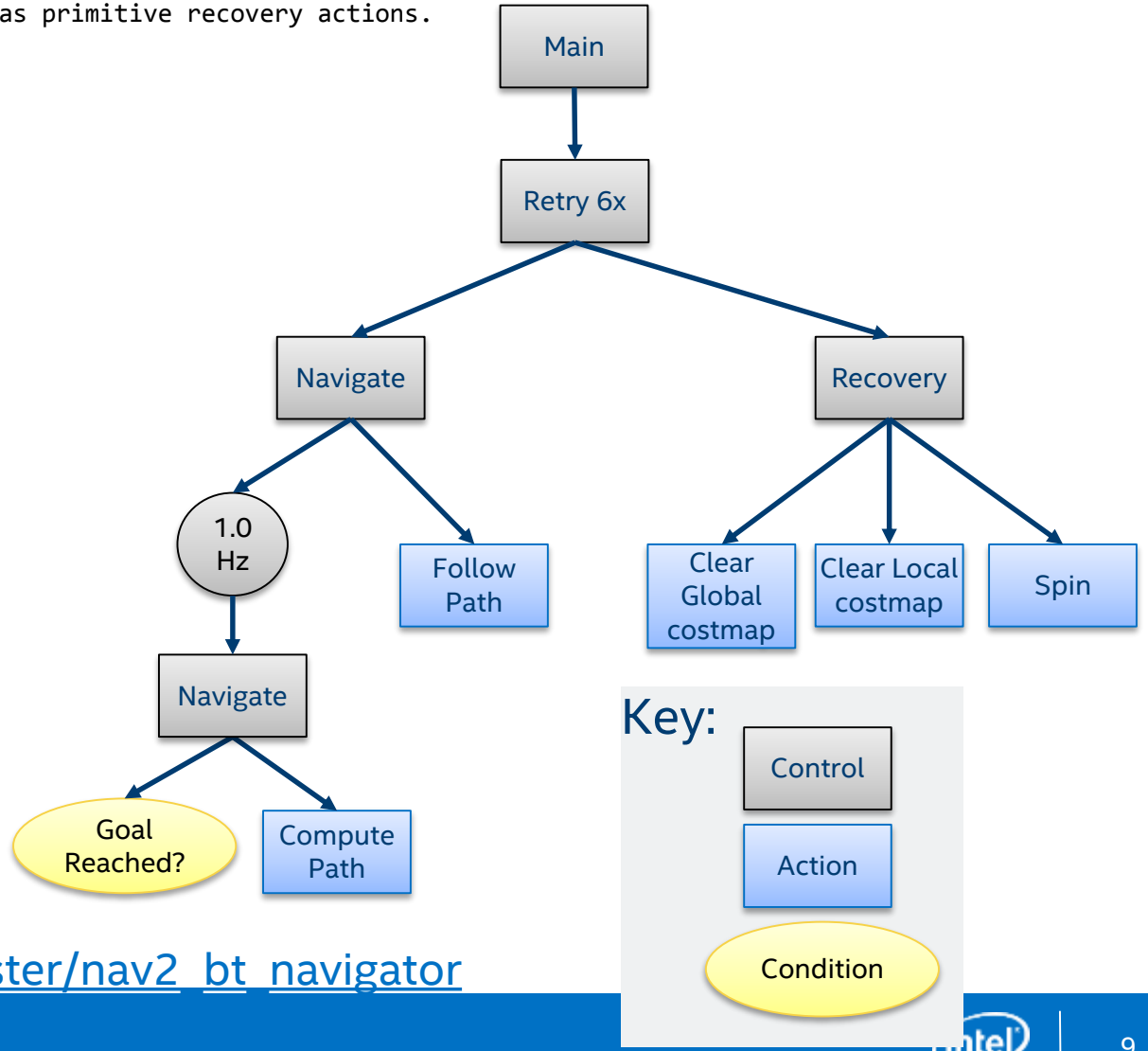
Program flow control decision trees, similar to state machines but hierarchical in nature



https://www.behaviortree.dev/

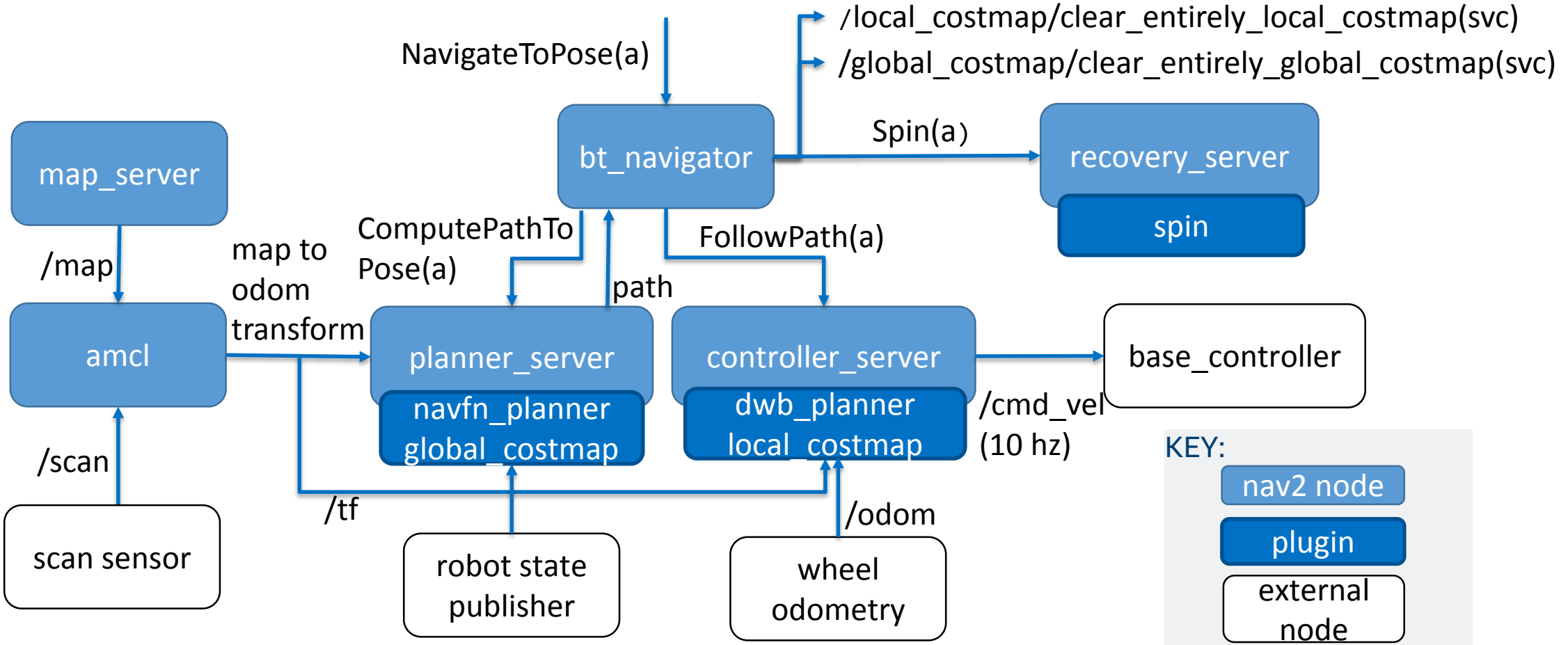
Behavior Tree XML example

```
<!--
  This Behavior Tree replans the global path periodically at 1 Hz and it also has primitive recovery actions.
-->
<root main_tree_to_execute="MainTree">
  <BehaviorTree ID="MainTree">
    <RecoveryNode number_of_retries="6">
      <Sequence name="NavigateWithReplanning">
        <RateController hz="1.0">
          <Fallback>
            <GoalReached/>
            <ComputePathToPose goal="${goal}" path="${path}"/>
          </Fallback>
        </RateController>
        <FollowPath path="${path}"/>
      </Sequence>
      <SequenceStar name="RecoveryActions">
        <clearEntirelyCostmapServiceRequest
              service_name="/local_costmap/clear_entirely_local_costmap"/>
        <clearEntirelyCostmapServiceRequest
            service_name="/global_costmap/clear_entirely_global_costmap"/>
        <Spin/>
      </SequenceStar>
    </RecoveryNode>
  </BehaviorTree>
</root>
```



https://github.com/ros-planning/navigation2/tree/master/nav2_bt_navigator

Navigation2 with Recovery



ROS2 Lifecycle nodes

Lifecycle nodes are ‘managed’ nodes that have an internal state machine

https://design.ros2.org/articles/node_lifecycle.html

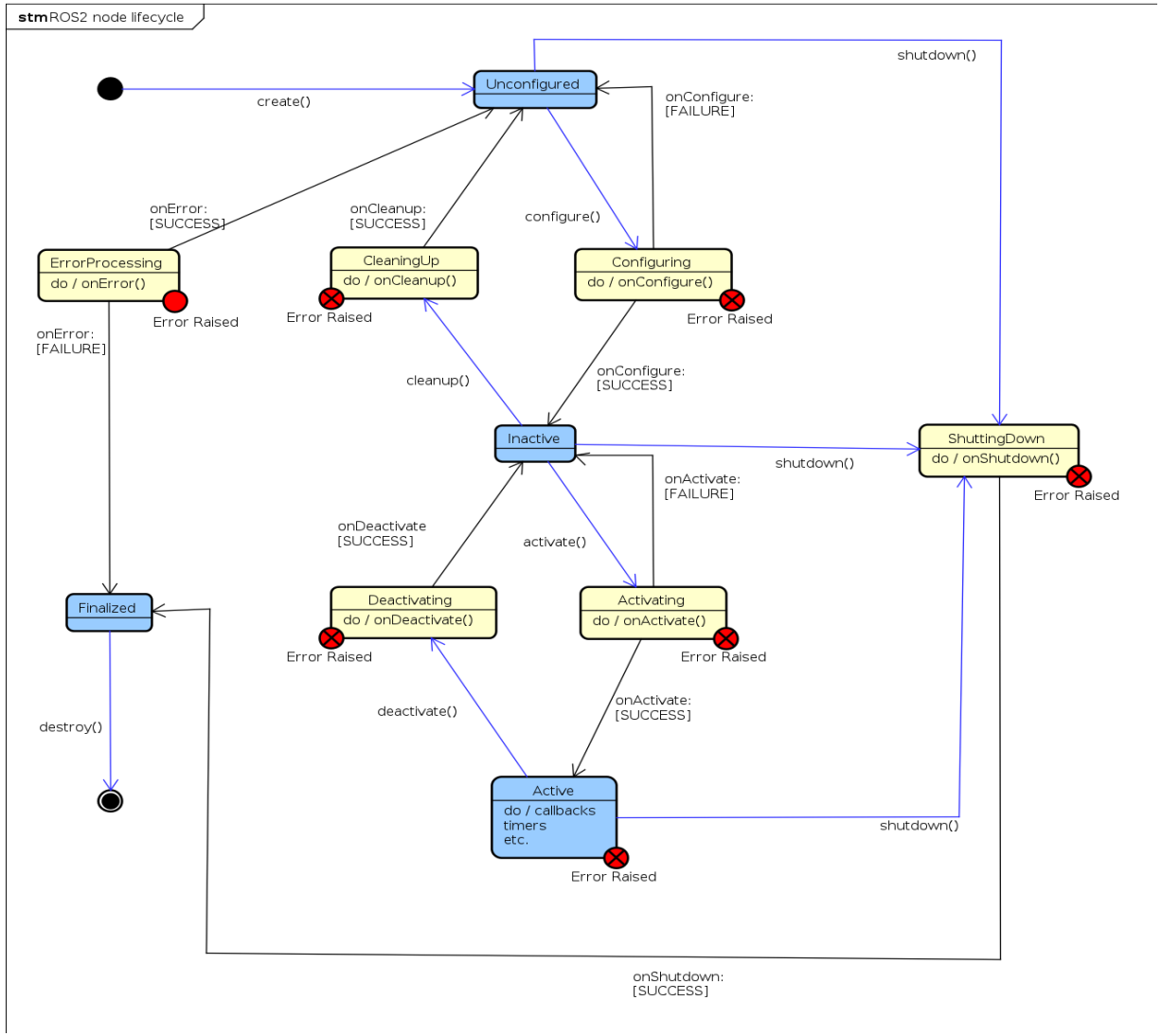
States:

- Unconfigured = created or new
- Inactive = ready to work
- Active = doing real work
- Finalized = ready to destroy

States are controlled through ‘change_state’ service

Each lifecycle node must implement the callbacks for the state transitions

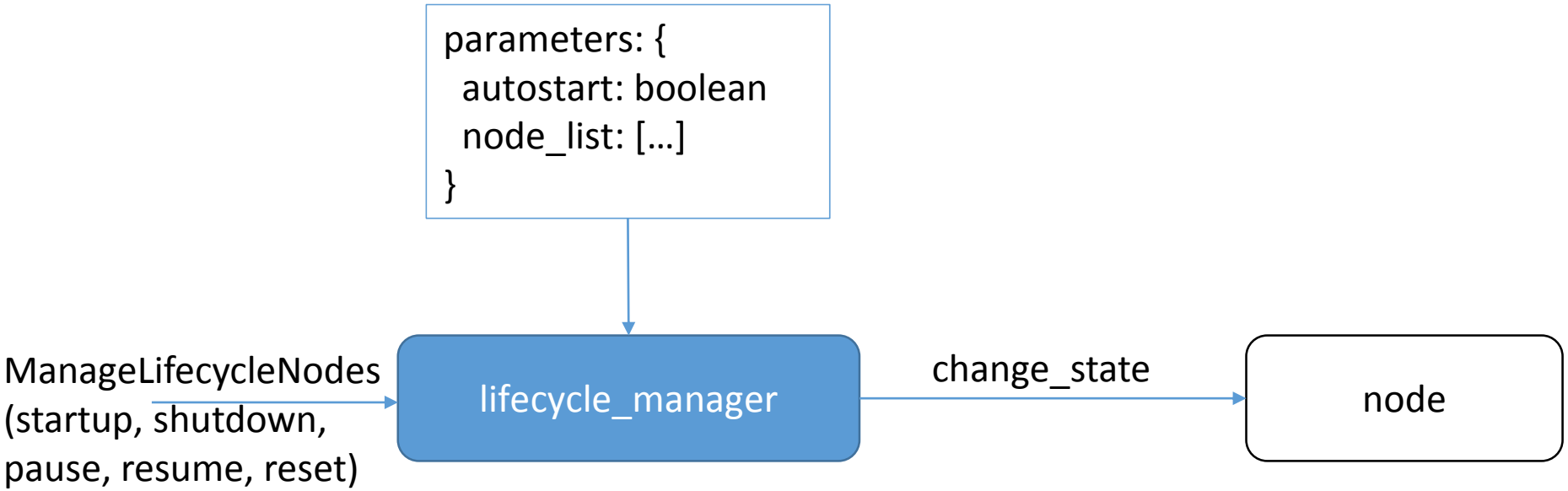
- onConfigure(), onActivate(), etc.



Navigation2 lifecycle manager

The lifecycle_manager node provides a ‘management’ service for controlling the startup and shutdown of the Navigation2 nodes

‘autostart’ parameter tells the lifecycle manager to start up everything in sequence automatically



Nav2 Plugin interface

The ‘nav2_core’ package contains abstract interfaces for plugins

- Global Planner – global_planner.hpp
- Local Planner – local_planner.hpp
- Recovery behaviors – recovery.hpp
- Goal checker – goal_checker.hpp
- Exceptions – exceptions.hpp

Navigation2 Bringup

nav2_bringup is a package within Navigation2 which provides the basic instructions and launch files for starting up the Navigation2 system

https://github.com/ros-planning/navigation2/tree/master/nav2_bringup

```
sudo apt install ros-dashing-navigation2 ros-dashing-nav2-bringup
source /opt/ros/dashing/setup.bash
# Launch the nav2 system
ros2 launch nav2_bringup nav2_bringup_launch.py use_sim_time:=True autostart:=True \
map:=<full/path/to/map.yaml>
```

For best results, follow the instructions on nav2_bringup/README.md

More tutorials and documentation is in progress, watch for updates

Simulation in the loop testing - nav2_system_tests

In ROS navigation, each pull request / code change was manually tested on a physical robot prior to being merged.

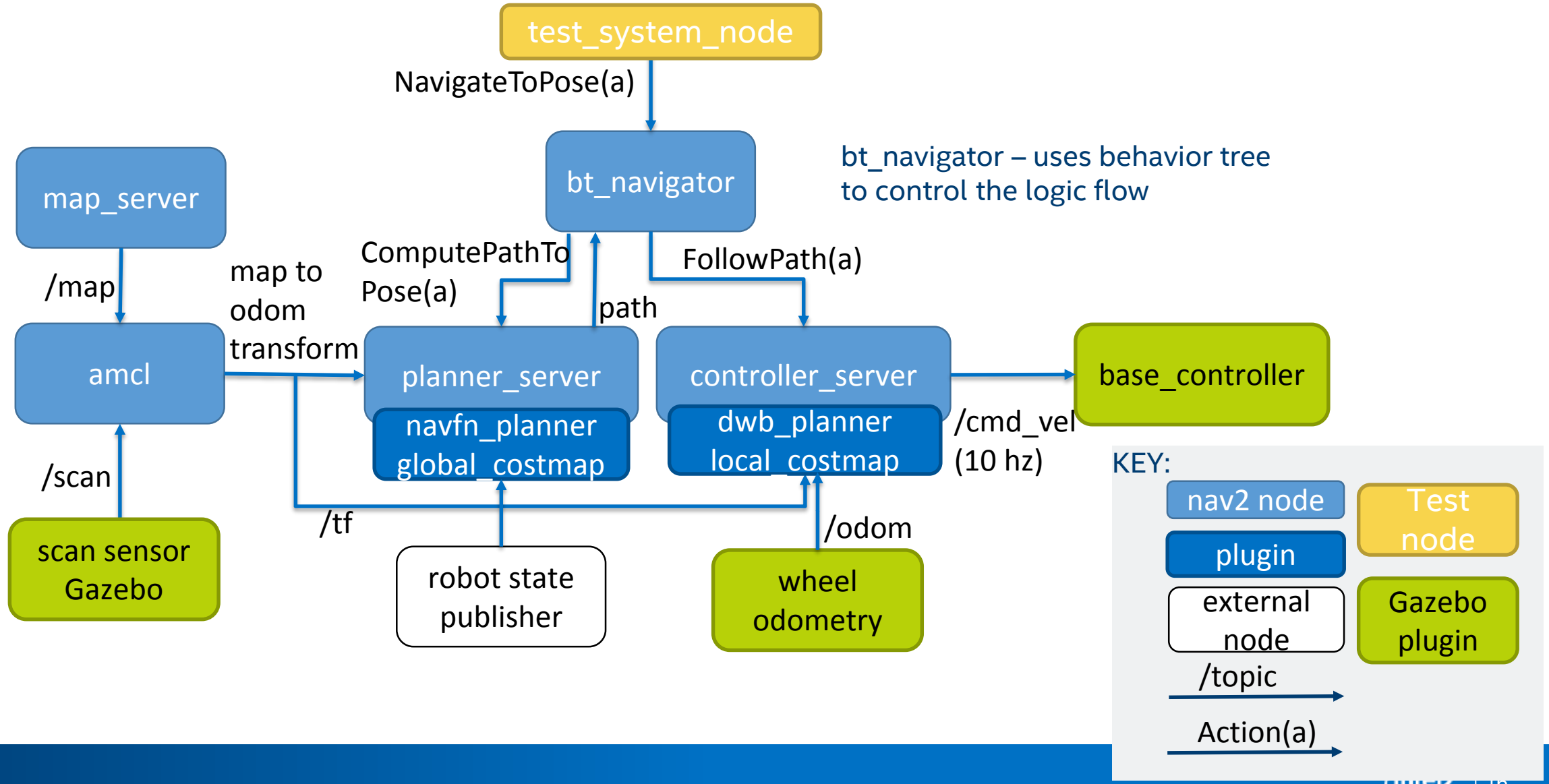
- **This is a time-consuming manual process**

By contrast during the development of navigation2, extensive testing was primarily done using Gazebo

To ensure **quality** and **maintainability**, an automated system test was created that uses Gazebo and a Turtlebot3 model to test that the system:

- Localizes correctly
- Successfully transitions into the ‘active’ state
- Navigates successfully to a known location

Navigation2 System Test



System test results

With the system test in place, able to find issues quickly (< 1minute to run)

- Example: prior to ROS2 Dashing release FastRTPS caused our test to break
- OSRF & Eprosima were able to reproduce the failures and fix

Able to run the test 100x/hour to find race conditions

- Drove pass rate from ~85% for Dashing to 95+% for Eloquent

Able to quickly test different DDS implementations for issues

- Found issue where CycloneDDS was initially failing more frequently than FastRTPS, ADLink was able to fix and increase to 95%+

System test is now integrated into ROS build farm “nightly” build

Summary

Navigation2 is a key component of ROS2 functionality

Navigation2 uses behavior trees, lifecycle nodes to provide customization and reliability, bringup launch files for ease of use, system tests for fast testing

Dashing release is available for ‘sudo apt install’

Eloquent release – coming by end of 2019, adds plugin support for all action servers (global planner, local controller, recoveries), multi-robot support and improved stability

Future Plans

Support ‘Timed Elastic Band’ as an additional local planner plugin

Release Nav2 packages for **ROS2 Eloquent**

Analyze and **improve system performance** metrics

Improve quality and robustness by improving test coverage

Increase **community involvement**

- Currently asking for input for F-turtle features

Build ROS2 expertise in academia

Continuously improve!

Navigation2 team

Matt Hansen, github: mkhansen-intel

Carl Delsey, github: crdelsey

Mike Jeronimo, github: mjeronimo

Carlos Orduno, github: orduno

Mohammad Haghighipanah, github: mhpanah

Brian Wilcox, github: bpwilcox

Melih Erdogan, github: mlherd

Yathartha Tuladhar, github: yathartha3

Steve Macenski, github: SteveMacenski

Special Thanks To

Robotis – Support and updates for Turtlebot3

Rover Robotics – Early adopters & contributors

Dan Rose, github: rotu

Yunji Robotics – Early adopters and demo partners (see our booth)

OSRF – for all their help and support

Ruffin White, github: ruffsl – Set up & maintains CircleCI

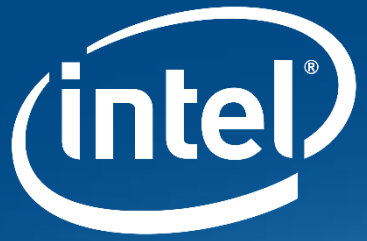
Call to Action

Try Navigation2!

- https://github.com/ros-planning/navigation2
- Submit issues and PRs

Participate in our ROS2 Working Group

- Navigation2 WG – Thursdays 3pm Pacific time
- https://groups.google.com/forum/#!forum/ros-navigation-working-group-invites
- Contact me if you have questions: discourse.ros.org - mkhansen



Thank You!