ENME480: Introduction To Robotics

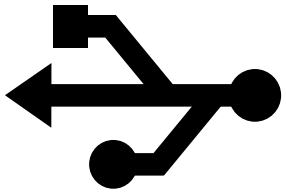
University of Maryland College Park

Introduction to ROS

Kaustubh Joshi
Alex Beyer

DEVELOPING A ROBOT

Turtlebot 2



Camera

PID Controller

Motors

Path Planning

SLAM

Object Tracking

Machine Learning

Data Logging

Error Analysis

ROBOTICS DEVELOPMENT

Components in Robot Development

Standard Robot Platforms

- Off the shelf robots with few sensors and mechanical components integrated
- E.g. Turtlebot, Nao, Husky, UR5

Sensors

- Equipment for perception
- E.g. Cameras, Proximity sensors, LiDAR, etc.

Electronics and Computing

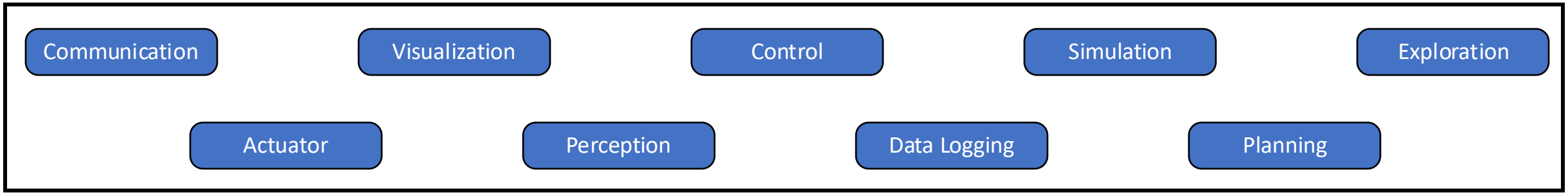
- Electronic components and computing and control hardware
- E.g. Motors, Encoders, Arduino, Raspberry Pi

Mechanical Equipment

- Additional mechanical hardware for task based requirements
- E.g. Manipulators, Grippers, etc.

ROADBLOCKS IN ROBOT DEVELOPMENT

Which modules are required in developing a robot?



Before ROS:

- If a new robot is to be made, all the modules will be required to developed again and needs to be done from scratch
- If any sensor needs to exchanged between two robots, entire code interface needs to be changed
- For adding a new functionality/module, entire code and framework needs to be changed

So, what does ROS do?

WHAT IS ROS

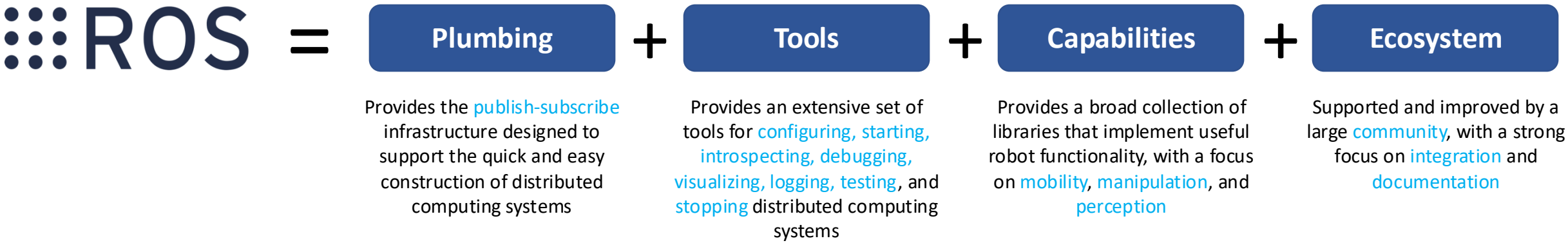
- Robot Operating System(ROS) is an open-source meta-operating system for robotics

How does it benefit robot development?

- It provides numerous basic algorithms for robotics for a variety of robot platforms (e.g. Turtlebot, UR5, etc.), sensors and electronic equipment
- Follows a modular approach which helps in using and building up on existing software and hardware according to user requirements
- Enables parallelization and networking, allowing processes to run on multiple systems (and robots)
- Independent of programming language (Python, C++, Java, etc.)
- Ready-to-use modules help in testing robots immediately
- In-built simulation (Gazebo) and visualization (RViz) environments – and now NVIDIA IssacSim too

ROS – AN OVERVIEW

What is ROS exactly? A middleware, a framework or an operating system?



What is the difference between ROS and an operating system?

- ROS sits on top of an OS and helps in running processes on a robot as a computer would.
- It does not natively provide the services which a traditional OS does.
  e.g. Memory Management, Web Browsers, Window Managers, etc.

Source: https://answers.ros.org/question/12230/what-is-ros-exactly-middleware-framework-operating-system/

CONCEPTS IN ROS

LEVELS OF CONCEPTS

COMPUTATION GRAPH

- Nodes
- Topics
- Messages
- Services
- Master
- Parameter Server
- Bags

FILESYSTEM

- Packages
- Metapackages
- Package Manifests
- Message Types
- Service Types

COMMUNITY

- Distributions
- Repositories
- ROS Wiki and Forums

NAMES

- Graph Resource Names
- Package Resource Names

CONCEPTS IN ROS

LEVELS OF CONCEPTS

COMPUTATION GRAPH

- **Nodes**
- **Topics**
- **Messages**
- **Services**
- **Master/Discovery**
- **Parameter Server**
- **Bags**

FILESYSTEM

- Packages
- Metapackages
- Package Manifests
- Message Types
- Service Types

COMMUNITY

- Distributions
- Repositories
- ROS Wiki and Forums

NAMES

- Graph Resource Names
- Package Resource Names

ROS NODE

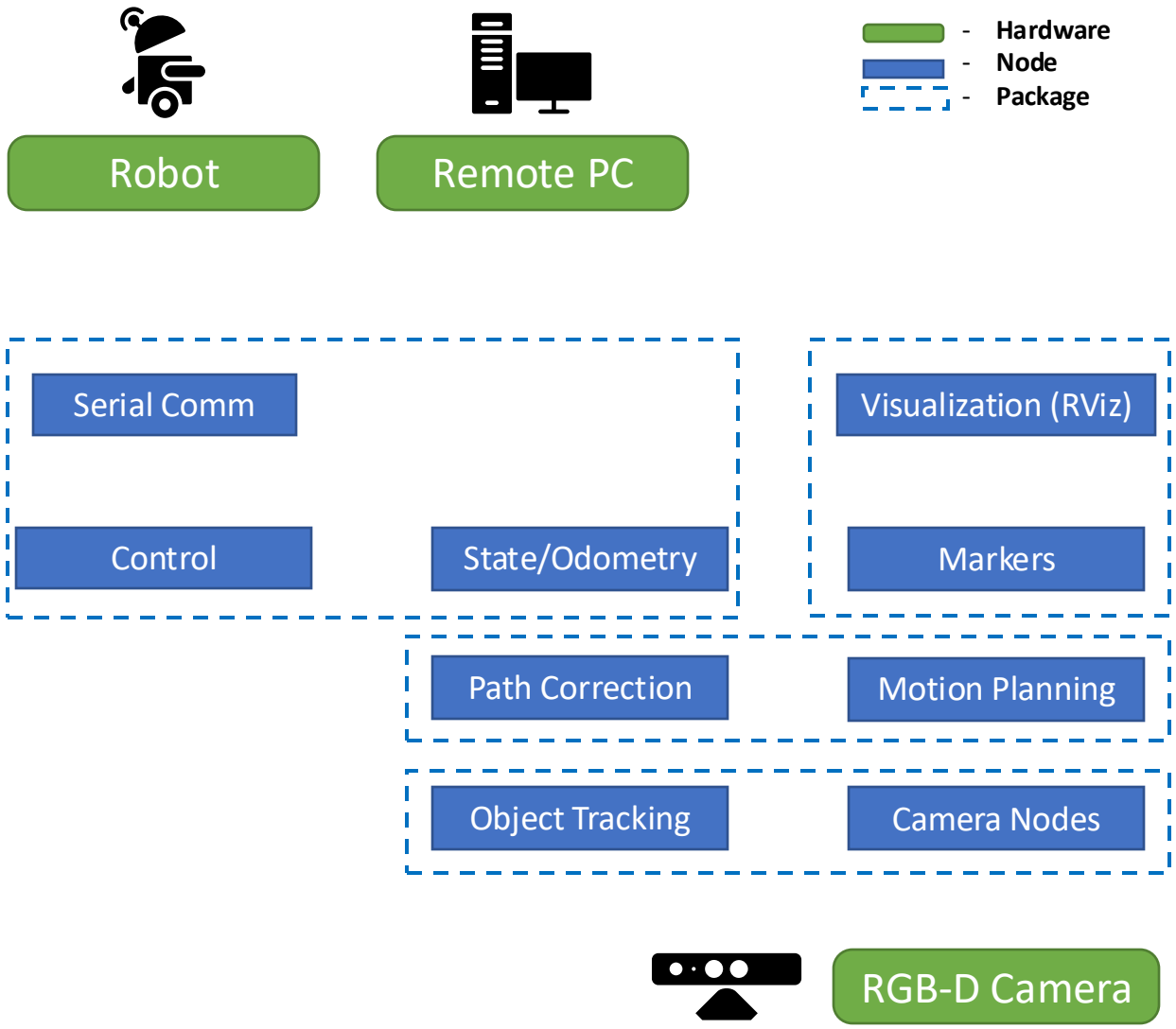
FEATURES

- Process that performs computation
  e.g. Sensor drivers, motor drivers, etc.
- Nodes communicate with each other by publishing or subscribing to **Topics**
- Nodes can offer or call a **Service** by sending a **Message**
- Written using ROS client libraries roscpp(C++) and rospy(Python)
- Command-line tool to display information:

```
rosnode  <argument>
ros2 node <argument>
```

- Have a **Graph Resource Name** and type which is **Package Resource Name**

EXAMPLE



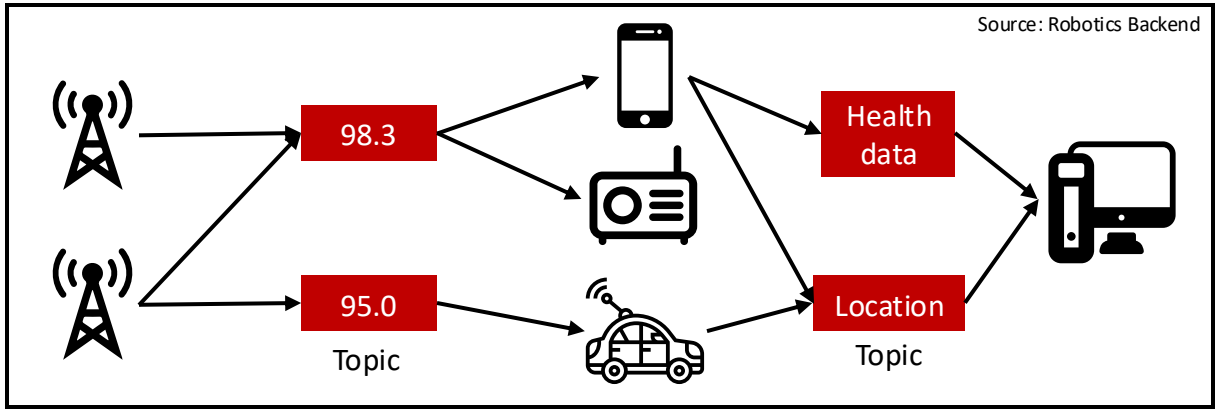
ROS TOPIC

FEATURES

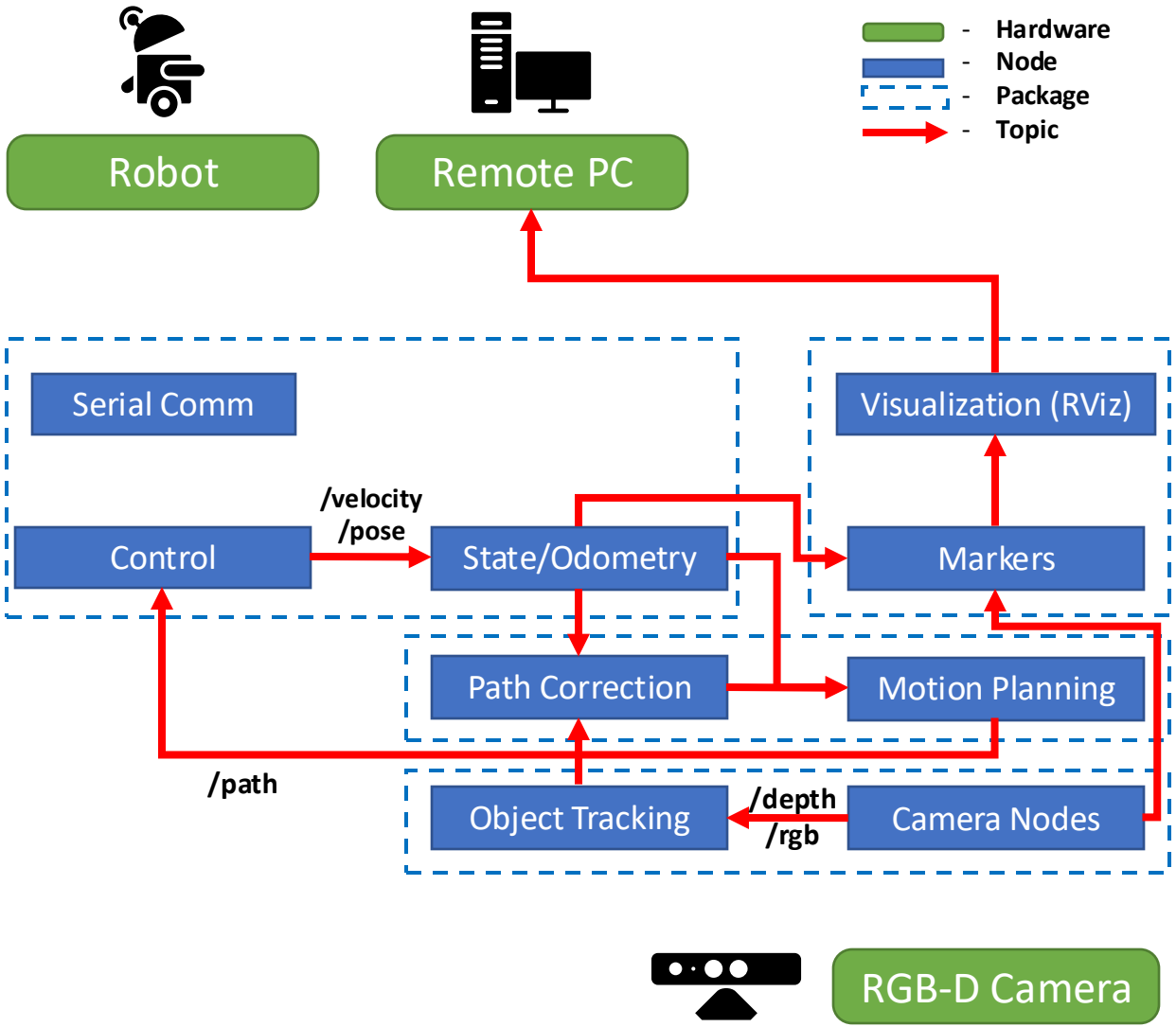
- Named buses over which nodes stream messages
- Nodes communicate with each other by publishing or subscribing to **Topics** *unidirectionally*
- Command-line tool to display information:

```
rostopic  <argument>
ros2 topic <argument>
```

- Publish/Subscribe Model: Many-to-many one way transport
  A node can publish and subscribe over multiple topics



EXAMPLE



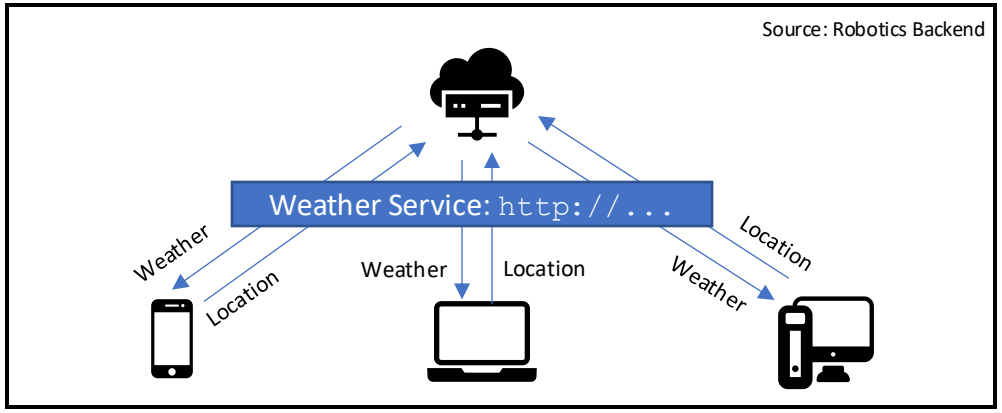
ROS SERVICES

FEATURES

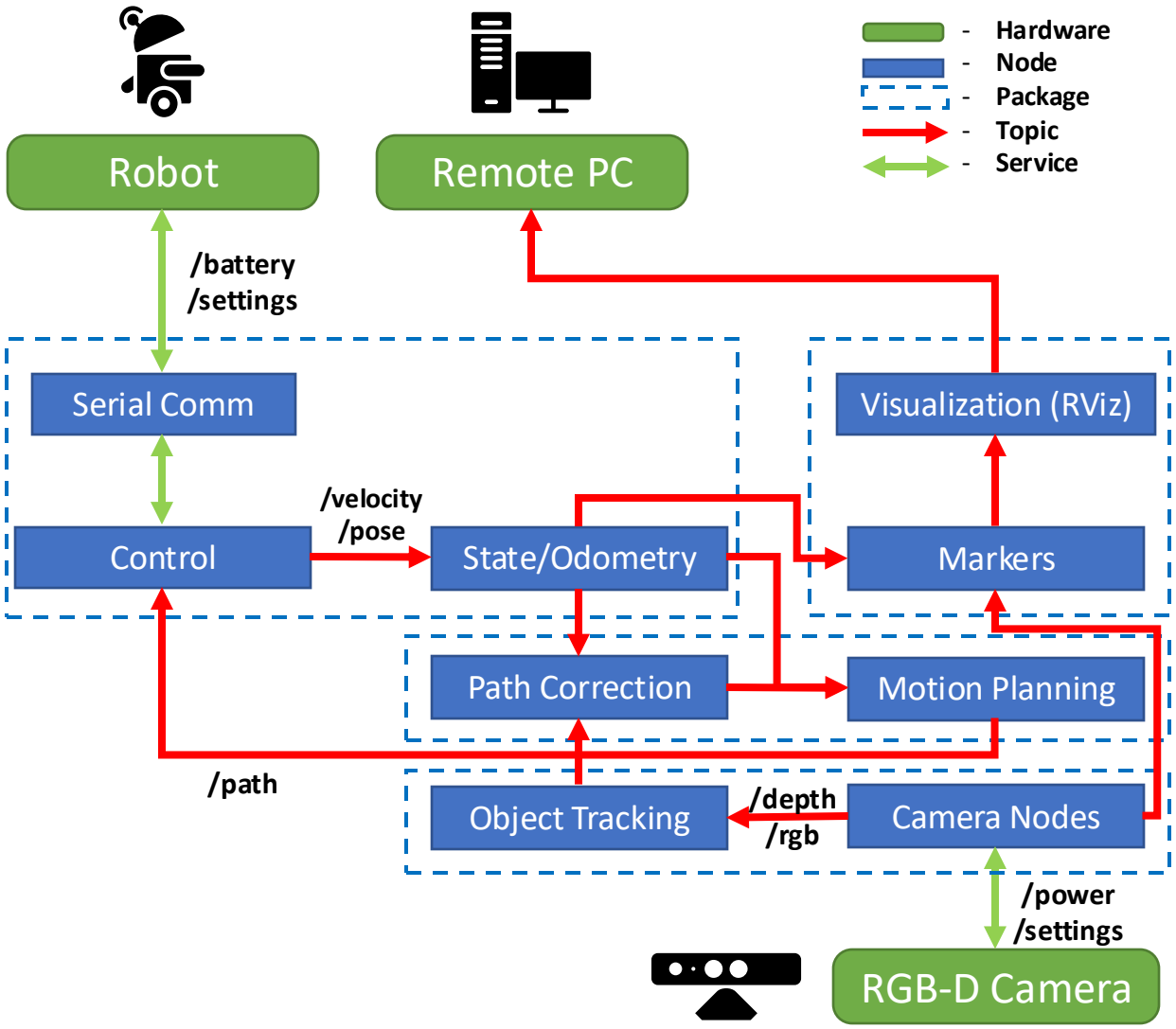
- Pair of messages between nodes; one for request and other for reply
- Request/Reply Model: one-to-one synchronous
- Used for computations and quick actions
- Command-line tool to display information:

```
rosservice or ros2 service <argument>
```

- Used when a client-server architecture is required



EXAMPLE



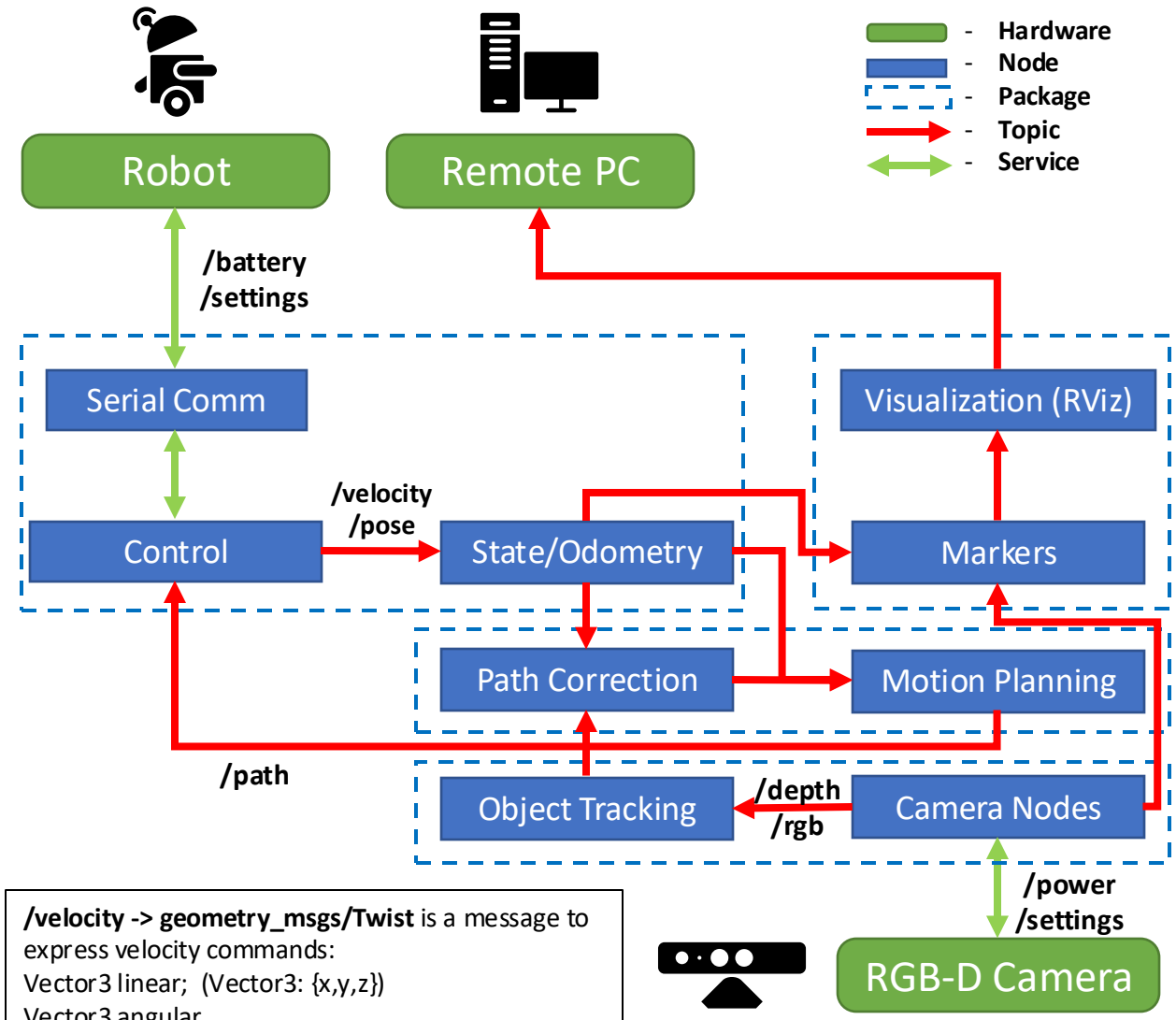
ROS MESSAGE

FEATURES

- Type of data structure to communicate between nodes
- Can include arbitrarily nested structures and arrays
- Can exchange a request or response as a **Service** call
- Command-line tool to display information:

```
rosmsg  <argument>
ros2 msg <argument>
```

EXAMPLE



/velocity -> geometry_msgs/Twist is a message to express velocity commands:
Vector3 linear; (Vector3: {x,y,z})
Vector3 angular

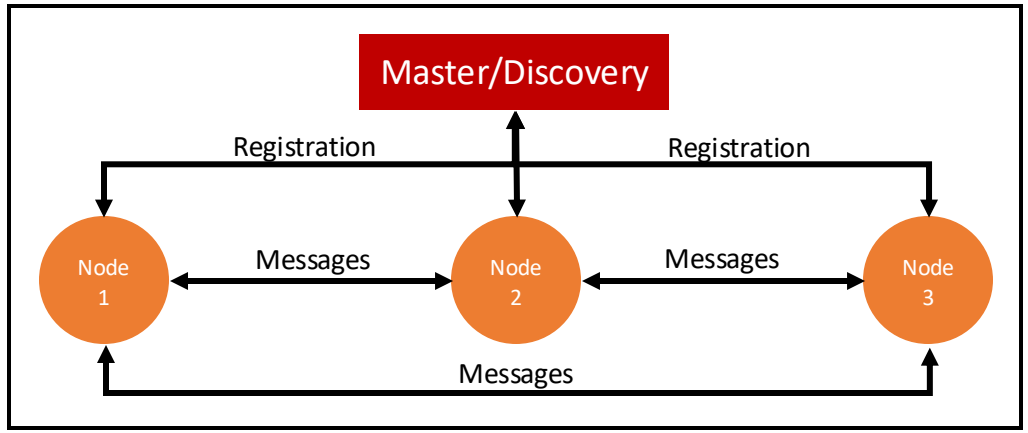
ROS1 MASTER / ROS2 DISCOVERY

FEATURES

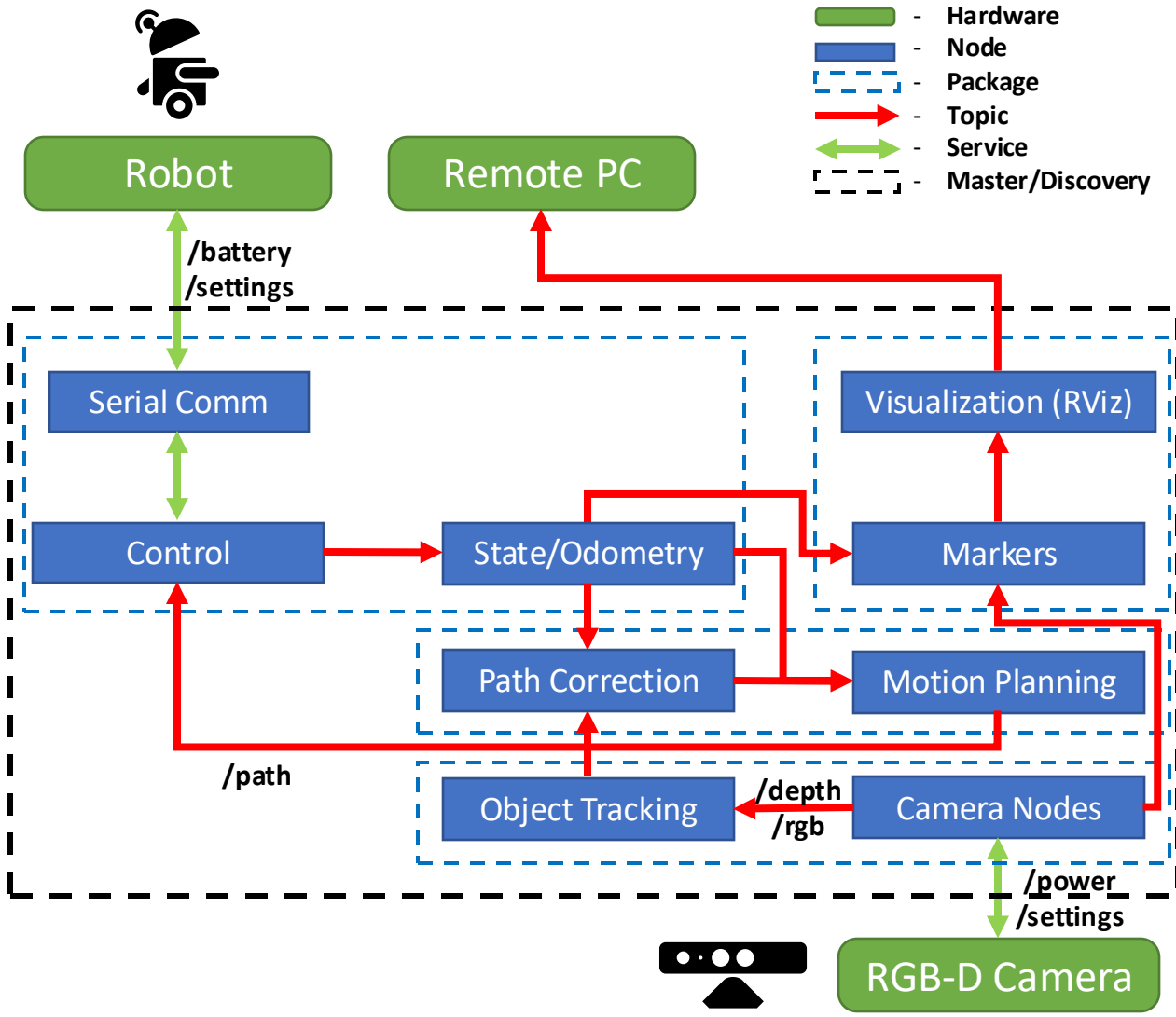
- Enables connection and transmission of messages for all the nodes in the system
- Tracks publishers and subscribers to all topics and services
- Command-line tool to initialise:

```
roscore
```

- No initialization required in ROS2 if a node is active



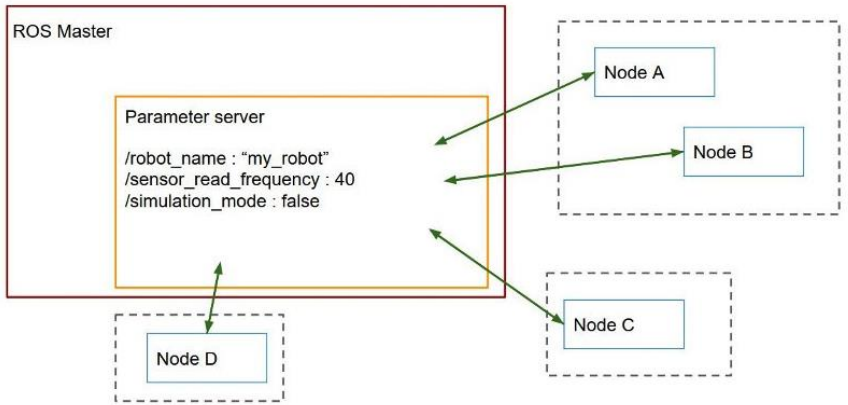
EXAMPLE



PARAMETER SERVER

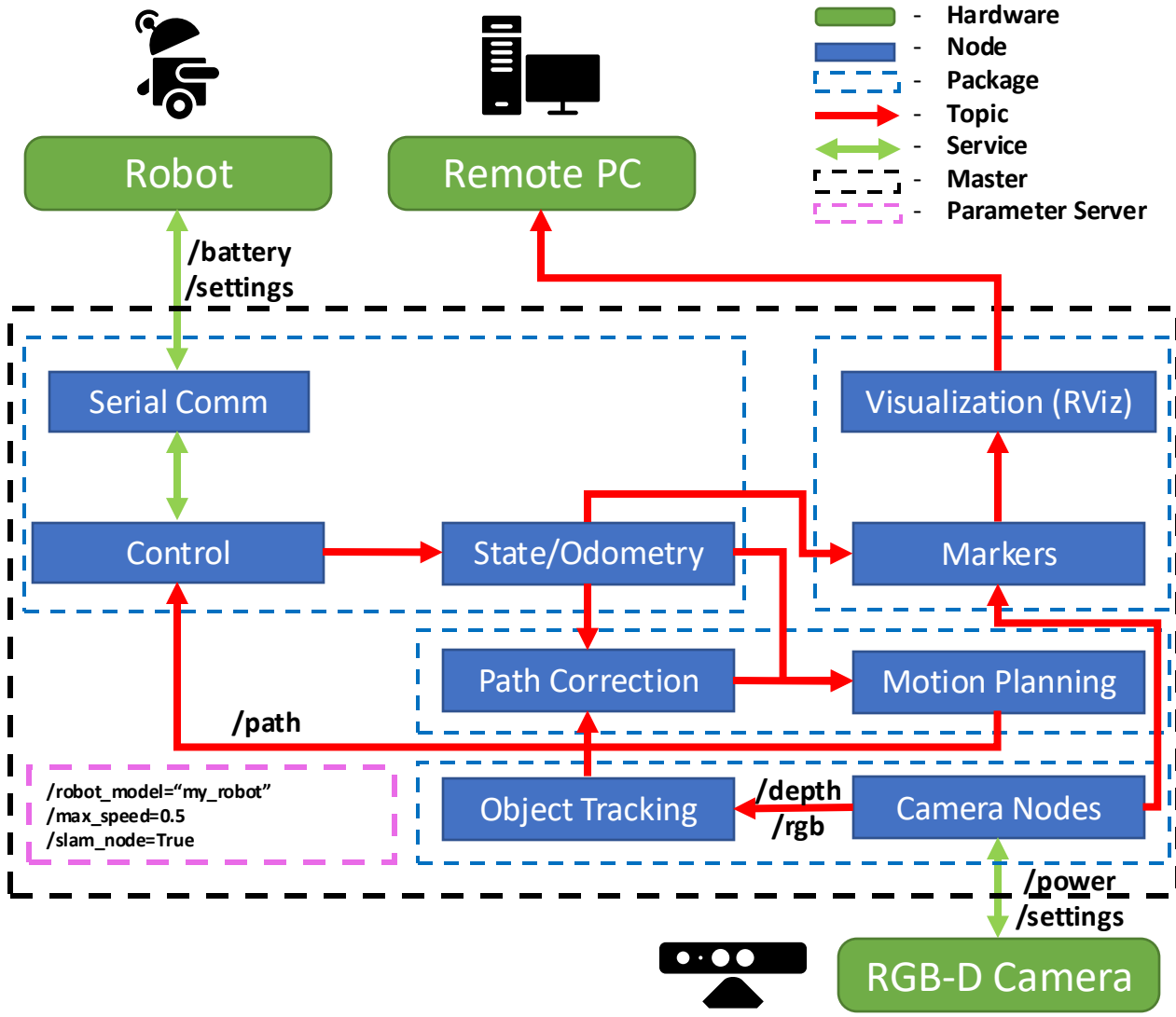
FEATURES

- Shared, multi-variate dictionary that is accessible through network APIs
- Runs inside ROS Master
- Nodes use this server to store and retrieve parameters at runtime
- Used for static, non-binary data such as configuration parameters



Source: Robotics Backend

EXAMPLE



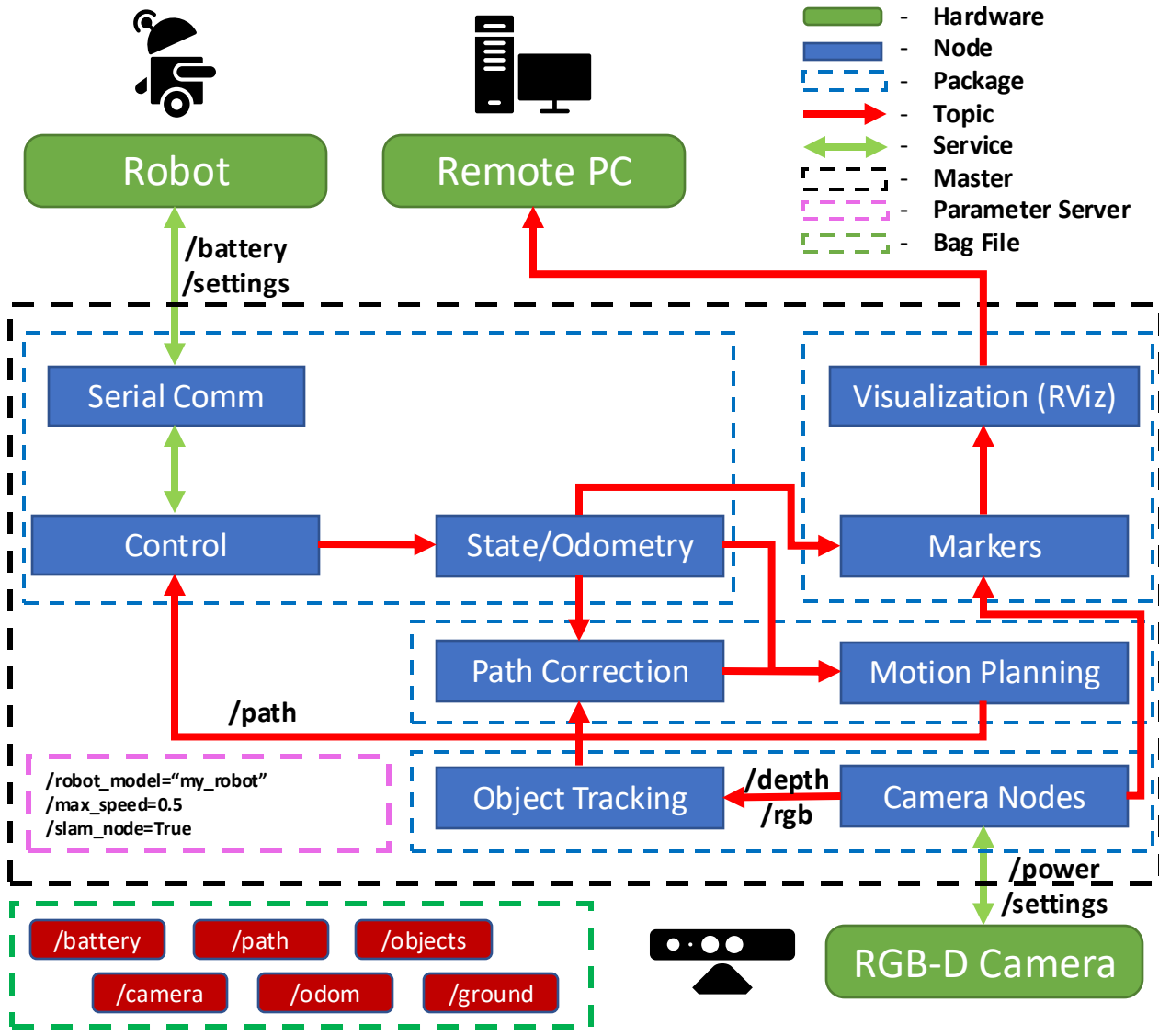
ROS BAGS

FEATURES

- Used for storing ROS Message data
- Bags subscribe to one or more ROS topics and store the serialized message data in a “bag”
  - ROS1 - `.bag` file
  - ROS2 - `.db3` file
- Can playback data as well
- Also provides the Parameter Server
- Command-line tools:

```
rosbag
ros2 bag
```

EXAMPLE



CONCEPTS IN ROS

LEVELS OF CONCEPTS

COMPUTATION GRAPH

- Nodes
- Topics
- Messages
- Services
- Master
- Parameter Server
- Bags

FILESYSTEM

- **Packages**
- **Metapackages**
- **Package Manifests**
- **Message Types**
- **Service Types**

COMMUNITY

- Distributions
- Repositories
- ROS Wiki and Forums

NAMES

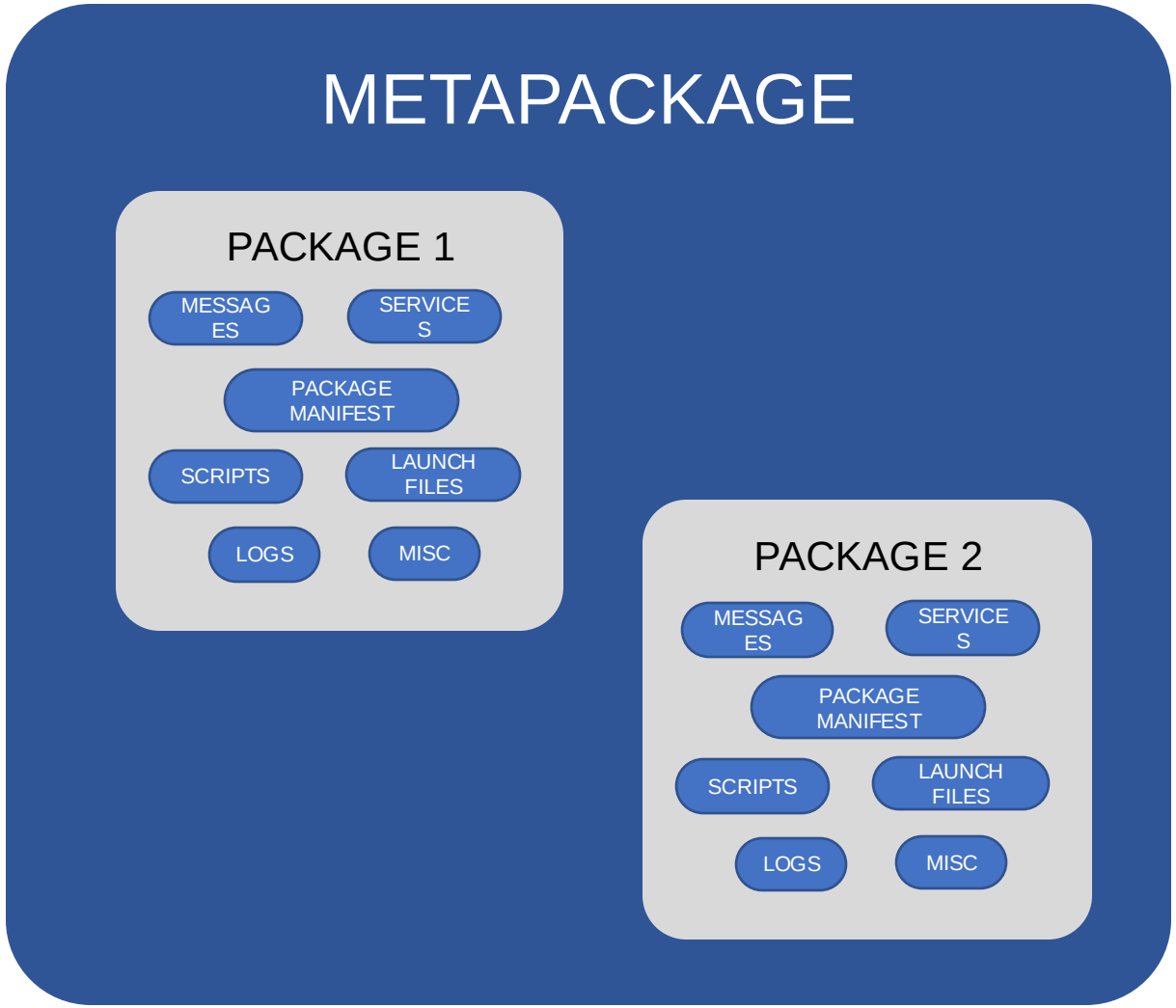
- Graph Resource Names
- Package Resource Names

ROS FILESYSTEM

FEATURES

- ROS software is organized in packages
- Contains nodes, configuration files, datasets, etc.
- Goal is to provide useful functionality in an easy-to-consume manner and reusability
- **Metapackages**: Specialized packages used as a virtual package. It references one or more related packages together
- **Package Manifest**: Mandatory XML file in a package which defines the properties of the package
- **Messages Description Files**: .msg files which describe the data values that ROS nodes publish
- **Service Description Files**: .srv files to enable request/response communication between ROS nodes

EXAMPLE



ROS PACKAGE

FEATURES

- ROS packages is contained in a **workspace**
- A package contains all codes and resources for a project

Source Space	Source code of the package is stored here. Can contain one or more packages
Build Space	This is where the package is compiled. Cache files and intermediate files are stored here
Devel Space	Built targets are placed here before being installed, for temporary usage
Install Space	Targets are installed here once they are built and is the final product

- Packages created using `colcon build`
- Creating a package tutorial: [Link]

EXAMPLE

```
ws/                              # Root Workspace
└── package_structure/           # Package Name
    ├── action/                  # custom ROS Action definitions
    ├── CHANGELOG.rst            # Compliant Changelog
    ├── CMakeLists.txt           # Compilation and Installation steps for C++ Code
    ├── config/                  # Configuration files
    ├── CONTRIBUTING             # Contribution Guidelines
    ├── doc/                     # Design or other documentation
    ├── include/                 # C++ Header files .hpp
    │   └── package_structure/   # Python Module. Keep the same name of package
    ├── launch/                  # Launch files
    ├── LICENSE                  # License file
    ├── models/                  # 3D Models (SDF)
    ├── msg/                     # Custom .msg files
    ├── package_structure/        # Python modules that can be imported to other packages
    ├── package.xml              # Define the properties of packages and dependencies
    ├── README.md                # Package purpose, installation steps, and Usage
    ├── rviz/                    # RVIZ Visualizer files
    ├── scripts/                 # Bash scripts
    ├── setup.py                 # Python Module installation steps
    ├── src/                     # C++ Source file .cpp
    ├── srv/                     # Custom service files
    ├── test/                    # Unit test files
    ├── urdf/                    # URDF Files
    └── worlds/                  # Gazebo world files
```

Ref: Medium Blog

CONCEPTS IN ROS

LEVELS OF CONCEPTS

COMPUTATION GRAPH

- Nodes
- Topics
- Messages
- Services
- Master
- Parameter Server
- Bags

FILESYSTEM

- Packages
- Metapackages
- Package Manifests
- Message Types
- Service Types

COMMUNITY

- **Distributions**
- **Repositories**
- **ROS Wiki and Forums**

NAMES

- Graph Resource Names
- Package Resource Names

ROS COMMUNITY

- **Distribution**: A version of ROS packages.

Linux Distribution	ROS 1 Distrbution (LTS)	ROS 2 Distribution (LTS)
Ubuntu 16.04 (Xenial Xerus)	Kinetic	N/A
Ubuntu 18.04 (Bionic Beaver)	Melodic	Dashing
Ubuntu 20.04 (Focal Fossa)	Noetic	Foxy, Galactic
Ubuntu 22.04 (Jammy Jellyfish)	No longer supported	**Humble**, Iron
Ubuntu 24.04 (Noble Numbat)	No longer supported	Jazzy

- **Repositories**: Software developed by independent institutions or individuals. Generally hosted on GitHub
- **ROS Wiki**: Documentation and tutorials for ROS and suitable packages
- **ROS Forum**: Website for Q&A related to ROS
- Others include a ROS Blog for news, ROS Mailing List for new updates, a Bug Ticketing System

CONCEPTS IN ROS

LEVELS OF CONCEPTS

COMPUTATION GRAPH

- Nodes
- Topics
- Messages
- Services
- Master
- Parameter Server
- Bags

FILESYSTEM

- Packages
- Metapackages
- Package Manifests
- Message Types
- Service Types

COMMUNITY

- Distributions
- Repositories
- ROS Wiki and Forums

NAMES

- **Graph Resource Names**
- **Package Resource Names**

NAMES

Graph Resource Names

- Provide a hierarchical naming structure for all resources in ROS Computation Graph (nodes, topics, messages, etc.)
- e.g. `/slam`
  `/turtlebot_teleop`
  `/map`

Package Resource Names

- Used for referring to the ROS Filesystem level concepts (message types, service types, etc.)
- E.g. `std_msgs/String` is an abbreviation for `/<path>/std_msgs/msg.String.msg`

ROS TOOLS AND COMMANDS

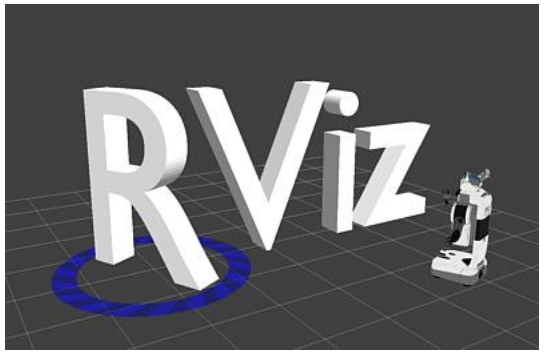
GAZEBO

- A 3D simulator for robotics applications
  `gazebo`
- Models can be imported and sensor plugins can be incorporated into the simulation environment in SDF or URDF format
- Environmental simulations can also be carried out for gravity, wind speeds, friction etc.

- Tutorials and Documentation: [Gazebo]

RViz

- 3D Visualization environment for ROS to let us know what is the robot thinking, seeing and doing
  `ros2 run rviz rviz`
- Different topics and messages can be visualized in the form seen or perceived by the robot



ROS TOOLS AND COMMANDS

COMMANDS

- Starting a node:
  `ros2 run <package_name> <node_name>`
- List of initialized nodes:
  `ros2 node list`
- List of available topics:
  `ros2 topic list`
- Get all messages published on a topic:
  `ros2 topic echo <topic_name>`
- Publishing messages through terminal line:
  `ros2 topic pub -r <rate-in-hz> <topic_name> <message_type> <message_content>`

COMMANDS

- System Visualization of Topics, Nodes, etc:
  `ros2 run rqt_graph rqt_graph`
- Live plot of data published on topics:
  `ros2 run rqt_plot rqt_plot`
- Console to display logging data
  `ros2 run rqt_console rqt_console`
  `ros2 run rqt_logger_level rqt_logger_level`
- Start multiple nodes in sequence with required parameters:
  `ros2 launch <package_name> <launch_file>`
- Recording data:
  `ros2 bag record -a` (All topics)
  `ros2 bag record -O subset <topic>` (Subset)