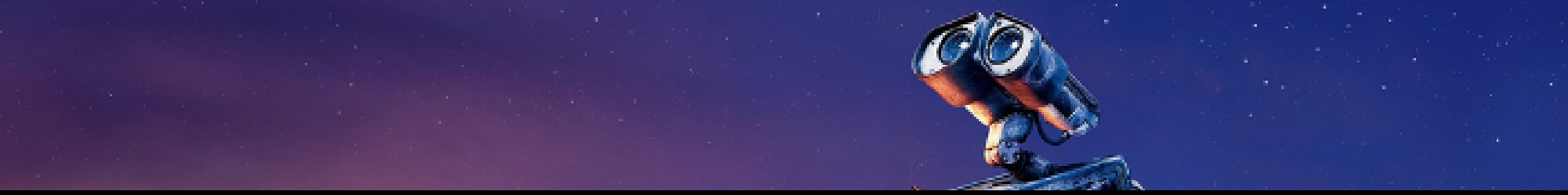


UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

Software for Robotics

RO51 - Introduction to Mobile Robotics

Zhi Yan

May 19, 2022

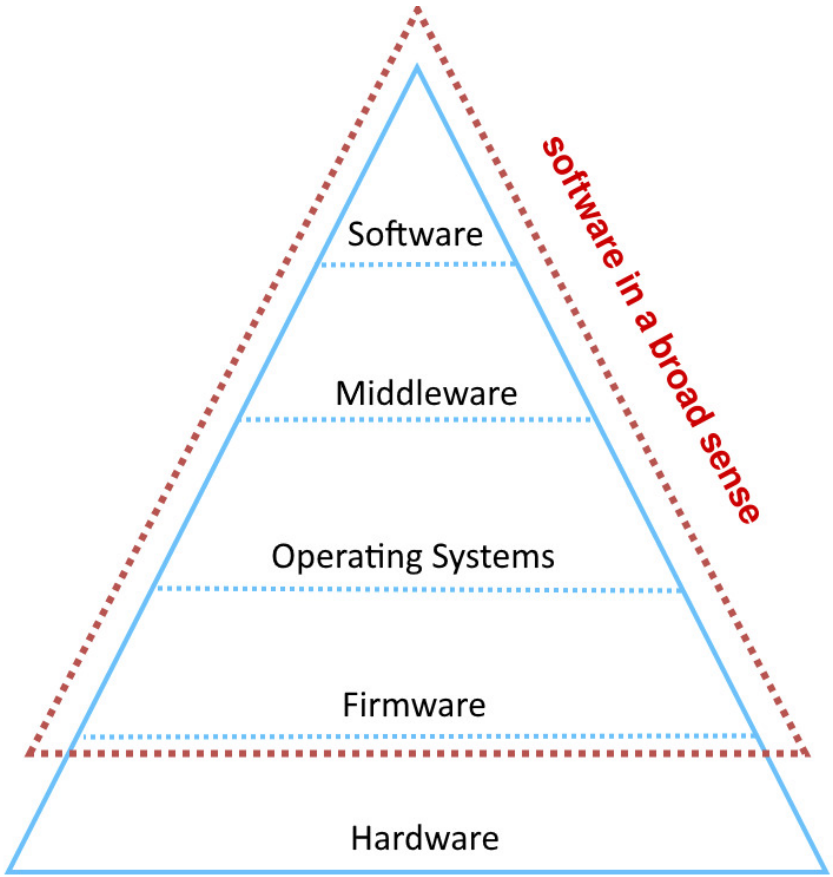
https://yzrobot.github.io/

www.utbm.fr

utbm
université de technologie
Belfort-Montbéliard

Context: computer

- A **computer system** (or computing device) includes: hardware, firmware, operating system (the "main software"), middleware and software.



Context: computer

- The pyramid structure means that more complex functionality can be written in fewer lines of code as you go up the layers.
- An advantage of this layered structure is that the software you write can run on different types of hardware without changing much in the software layer itself.
- But as you go up the layers, it becomes harder to know what is happening down at the hardware level.

Context: computer

- Hardware: CPU, RAM, motherboard, etc.
- Firmware (stored in ROM): BIOS, OpenCR firmware, etc. => Taking care of the hardware related dependencies.
- Operating System (OS): Microsoft Windows, macOS, Linux, etc.
- Middleware: Tomcat, Player Project, ROS, etc. => Bridging the gap between an OS or database and applications.
- Software: Minecraft, Photoshop, Firefox, etc.

Robotics middleware

Representative **open-source** middleware in the field of mobile robots:

- CARMEN (Carnegie Mellon Robot Navigation Toolkit)[1]
  - Not strictly middleware, but has great significance and influence on the mobile robotics community.
- Player (*"All the world's a **stage**, And all the men and women merely **players**."* - William Shakespeare, As You Like It)[2]
  - Player can be seen as the past life of ROS.
- ROS (Robot Operating System)[3]
  - The *de-facto* standard for advanced robotic systems today.

[1] `http://carmen.sourceforge.net/`
[2] `http://playerstage.sourceforge.net/`
[3] `https://www.ros.org/`

CARMEN

- Active years: 2003 - 2008
- Software organization: modularity
- Communication between programs: handled using IPC (Inter-Process Communication) sockets
- Tools: process monitoring and debugging
- Underlying OS: Linux (Red Hat, SUSE)
- Programming language: written in C
- Code hosting: SourceForge
- License: GNU General Public License

CARMEN

- Supported robot platforms: iRobot (ATRV, B21R), ActivMedia (Pioneer I & II), etc.
- Supported sensors: 2D LiDAR, Sonar and GPS
- Main components:
  - Path planning module
  - Localization module
  - Scan-matching and mapping module
  - Message logging and playback functionality
  - Centralized parameter server

Player

- Active years: 2000 - 2010
- Software organization: modularity
- Communication between programs: handled using TCP (Transmission Control Protocol) sockets
- Underlying OS: Linux, Unix-like (Solaris, *BSD, Mac OSX), Windows.
- Programming language: mainly written in C/C++
- Code hosting: SourceForge
- License: GNU General Public License

Player

- Supported robot platforms: Pioneer, iRobot, Segway, LEGO, and more
- Supported sensors: 2D LiDAR, Sonar, GPS, camera, and more
- Main components:
  - `libplayercommon (C)`: error reporting facilities
  - `libplayercore (C++)`: basic messaging and queueing functionality, support for loading plugins, parsing configuration files
  - `libplayerdrivers (C++)`: the drivers that are included with the Player distribution (and which were compiled)
  - `libplayertcp (C++)`: support for TCP client/server transport
  - `libplayerinterface (C++)`: support for interface parsing and XDR (External Data Representation) data marshaling

ROS

- Active years: 2007 - present
- Software organization: (high) modularity
- Communication between programs: handled using XML-RPC (XML Remote Procedure Call) (for "Master") and TCP/UDP (User Datagram Protocol) (for "Topic") sockets
- Underlying OS: Linux, Unix-like (Solaris, *BSD, Mac OSX), Windows.
- Programming language: mainly written in C++ and Python
- Code hosting: GitHub
- License: 3-clause BSD License (for core of ROS)

ROS

- Supported robot platforms: many
- Supported sensors: many
- Main components:
  - Communication infrastructure (master, publisher, subscriber, etc.)
  - Robot specific features (*_msg, tf, urdf, actionlib, amcl, gmapping, navigation, etc.)
  - Tools (command-line tools, rviz, rqt, etc.)
- Powerful support (integration with other libraries):
  - Gazebo
  - OpenCV
  - PCL
  - MoveIt!
  - etc.

ROS

Why ROS?

- A distributed, modular design
- A vibrant community
- Permissive licensing
- A collaborative environment

Recall the ROS concepts (c.f. Lecture 1):

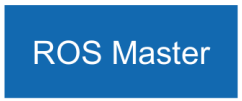
- The **filesystem** level
- The **computation graph** level <= Many "standards" are defined, which greatly improves the reusability of software.
- The **community** level

ROS computation graph

ROS master:

- Manage the communication between *nodes*.

Start a *master* with: `$ roscore`

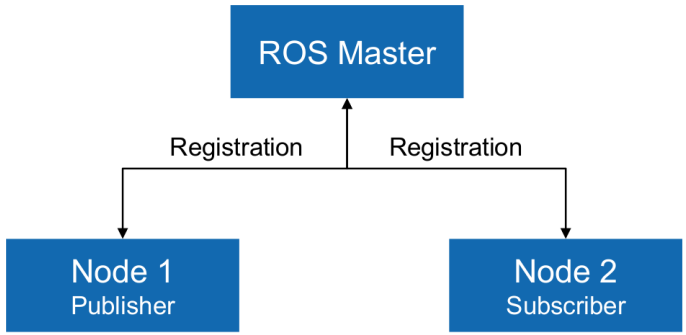


ROS computation graph

ROS nodes:

- Every *node* registers itself at startup with the *master*.
- Single-purpose, executable program.
- Individually compiled, executed and managed, organized in *packages*.

Launch a *node* with: `$ rosrun package_name node_name`

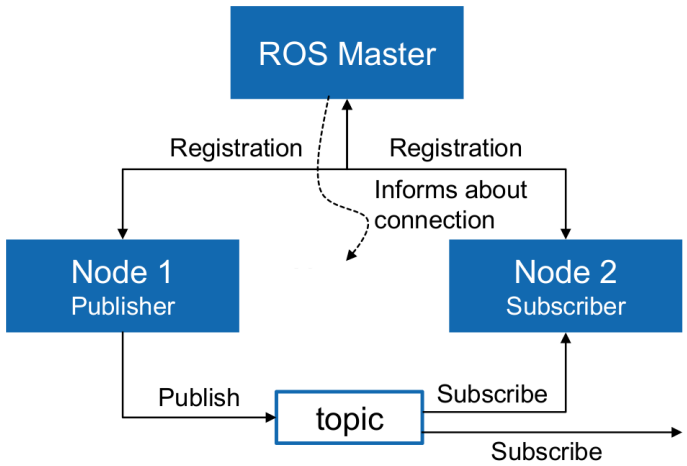


ROS computation graph

ROS topics:

- The *topic* is a name used to identify the content of *messages*.
- *Nodes* communicate with each other over *topics*.
- A *node* can publish or subscribe to *topics*.

List currently active *topics* with: `$ rostopic list`

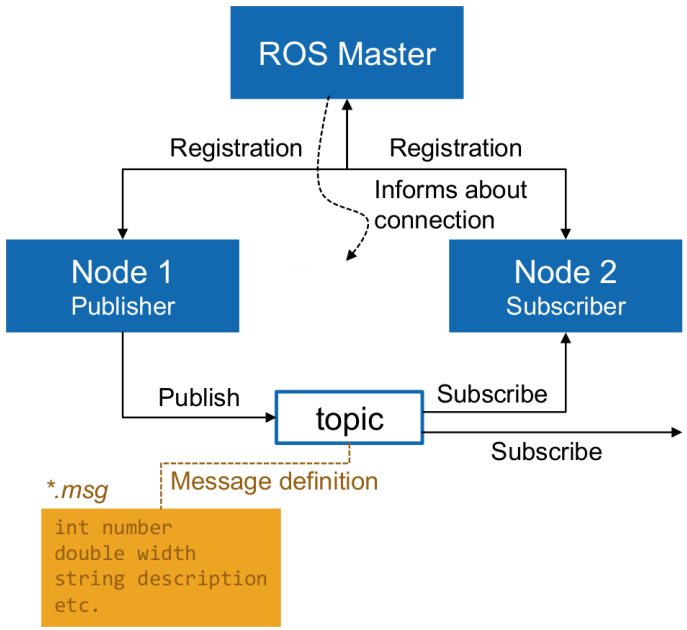


ROS computation graph

ROS messages:

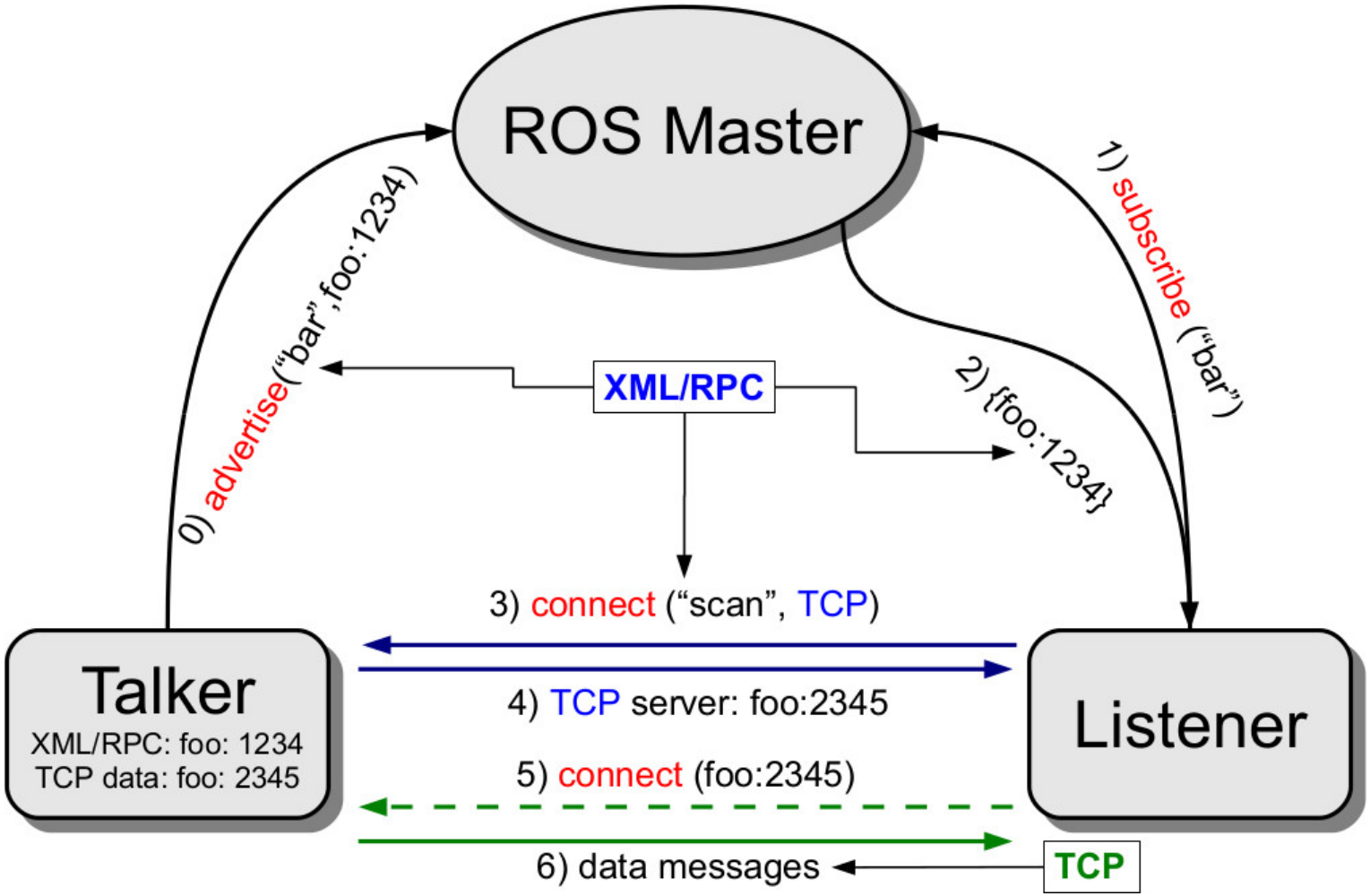
- A simple data structure, comprising typed fields.
- Standard primitive types (integer, float, etc.) are supported.
- Can include arbitrarily nested structures and arrays.

Show *message* content with: `$ rostopic echo topic_name`



ROS computation graph

A closer look at ROS topics (with communication details):

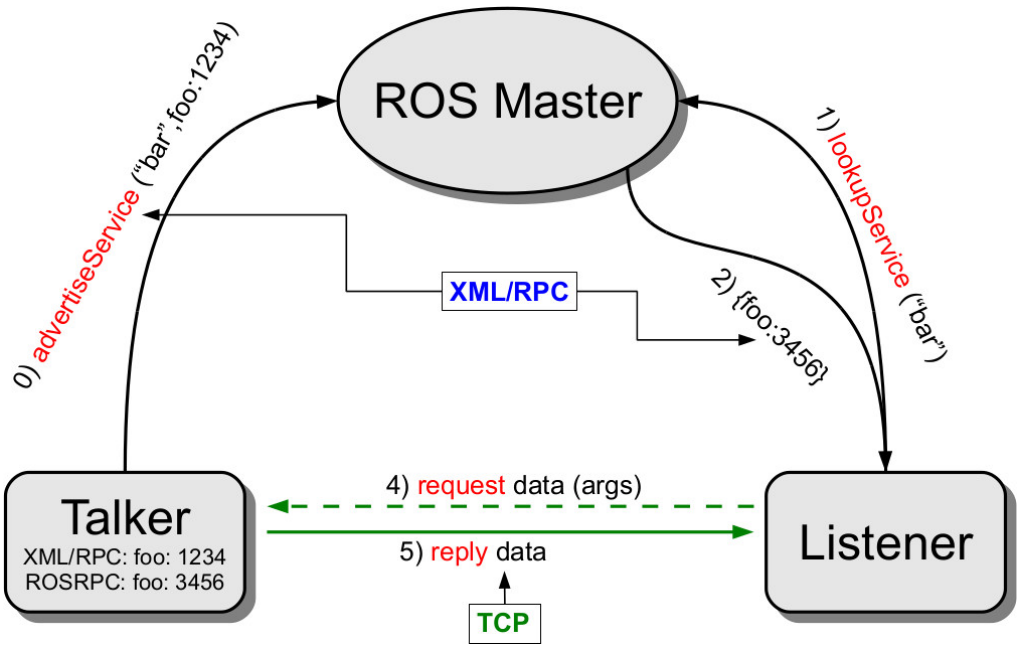


ROS computation graph

ROS services:

- Designed for RPC request / reply interactions.
- Implemented by a pair of *messages*: one for the request and one for the reply.

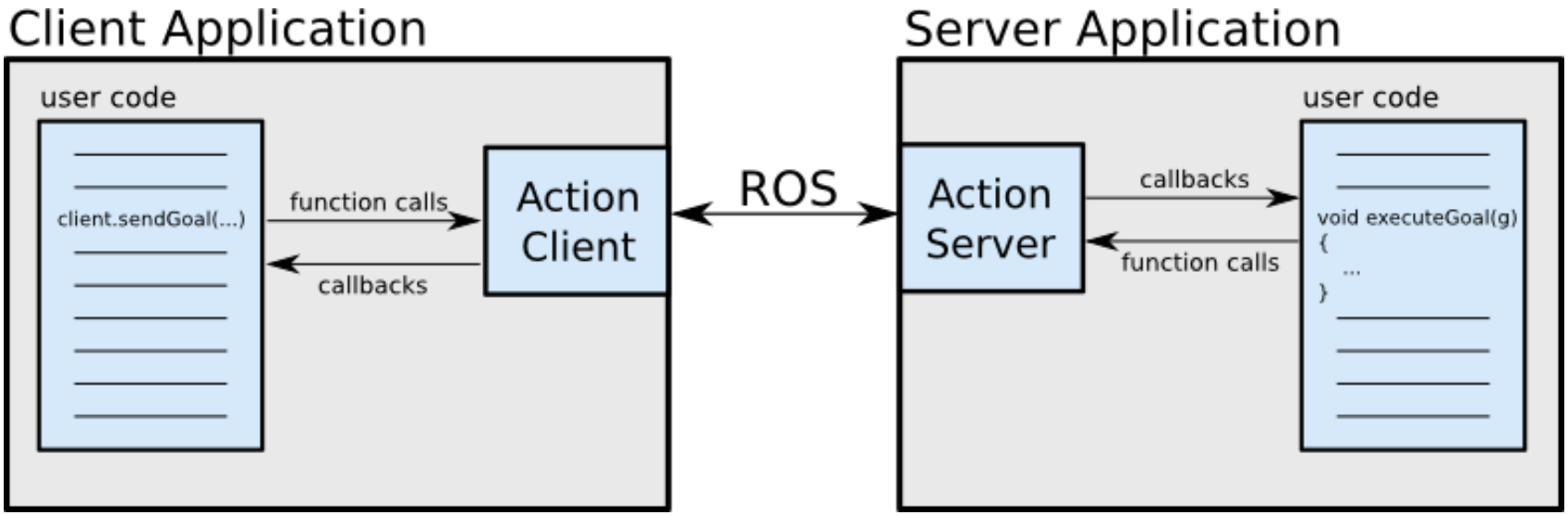
List currently active *services* with: `$ rosservice list`



ROS computation graph

ROS actions (i.e. *actionlib*):

- For services that take a long time to execute, *actionlib* allows the user to cancel the request during execution or get periodic feedback on the progress of the request.
- Different from getting "quick" response through ROS *services*.

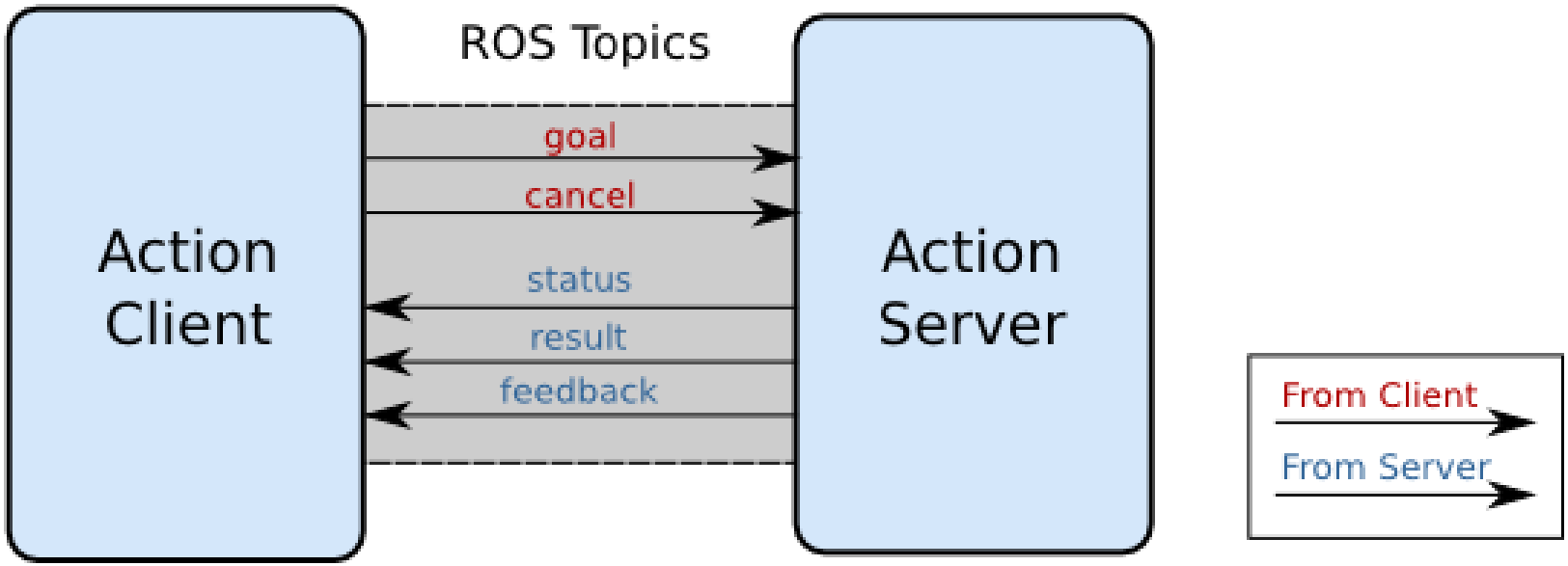


ROS computation graph

ROS actions (i.e. *actionlib*):

- *Action protocol* relies on ROS topics to transport messages.
- Default defined message types: *Goal*, *Feedback*, and *Result*.

Action Interface



ROS 2

- Active years: 2014 - present
- Under very active development, release cycle: twice a year (unlike every 1 to 2 years for mature ROS 1)
- Does not break ROS 1, nor does it rollout into ROS 1
- Breaking API with ROS 1, but conceptually very similar
- Building on DDS (Data Distribution Service (for Real-Time Systems))
- Interoperating with ROS 1 (by *ros1_bridge*)

ROS 2

Why ROS 2?

- Modern API, minimal dependencies, and better portability (e.g. small embedded platforms)
- Benefits of underlying DDS middleware:
  - Reliability QoS settings
  - UDP Multicast, shared memory, TLS over TCP/IP
  - Real-Time capable
  - Master-less discovery
  - Minimal dependencies
- Easier to work with multiple nodes in one process
- More dynamic runtime features like topic remapping and aliasing
- Lifecycle management and verifiable systems
- And more!

Robotics simulators

- A robotics simulator is a simulator used to create application for a **physical robot** without depending on the actual machine.
- Typically powered by physics engines.
- Benefits: saving (development and debugging) cost and time.
- Often, (one hopes) applications can be deployed onto the real robot without modifications.
- Covered in this lecture:
  - **Stage**[4]: a 2.5D robotics simulator
  - **Gazebo**[5]: a 3D robotics simulator

[4] http://playerstage.sourceforge.net/index.php?src=stage
[5] http://playerstage.sourceforge.net/index.php?src=gazebo

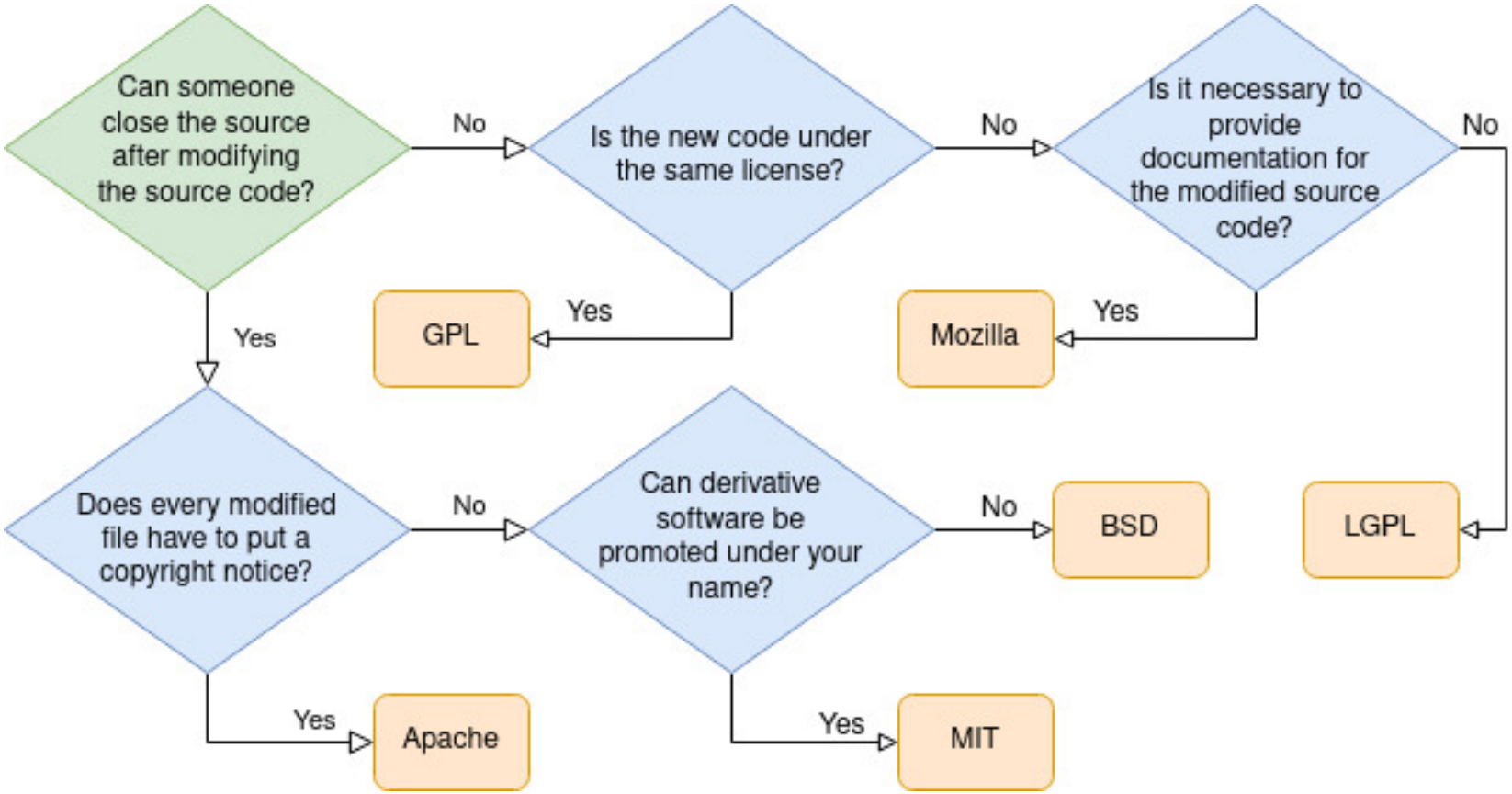
Stage

- Active years: 2000 - present
- Early used as a plugin for Player, subsequently provided support for ROS
- Ideal for rapid prototyping of real robots
- Code hosting: SourceForge, GitHub
- License: GNU General Public License
- A nice video here!

Gazebo

- Active years: 2003 - present
- Early used as a plugin for Player, subsequently became a standalone software and served as a standard simulator for ROS
- Ideal for realistic simulation of various robots and environments
- Code hosting: SourceForge, GitHub
- License: Apache 2.0

How to choose a free software license



Robotics datasets

- **Radish** (`http://radish.sourceforge.net/`): SLAM
- **FP7 project STRANDS** (`https://lcas.lincoln.ac.uk/nextcloud/shared/datasets/`): long-term robot autonomy
- **H2020 project ENRICHME** (`https://lcas.lincoln.ac.uk/wp/research/data-sets-software/lcas-thermal-physiological-monitoring-dataset/`): thermal physiological monitoring
- **H2020 project FLOBOT** (`http://lcas.github.io/FLOBOT/`): robot perception
- **EU Long-term** (`https://epan-utbm.github.io/utbm_robocar_dataset/`): autonomous driving

Open science

- Open Science is the movement to make scientific research and its dissemination accessible to all levels of an inquiring society, amateur or professional.
- Transparent and accessible knowledge shared and developed through collaborative networks.
- **Open access (research paper), open source (code), open dataset**, and more.
- European actions: FOSTER Open Science (`https://www.fosteropenscience.eu/`)
- National actions: Archive ouverte HAL (`https://hal.archives-ouvertes.fr/`)
- Regional actions: dat@UBFC (`https://data.ubfc.fr/datubfc/`)

Summary

- Robotics middleware: CARMEN, Player, ROS
- Robotics simulators: Stage, Gazebo
- Robotics datasets
- Open science

The end

Thank you for your attention!

Any questions?