Robot operációs rendszerek és fejlesztői ökoszisztémák

ROS alapok 2.

Gincsainé Szádeczky-Kardoss Emese

2025. szeptember 22.

KUKA

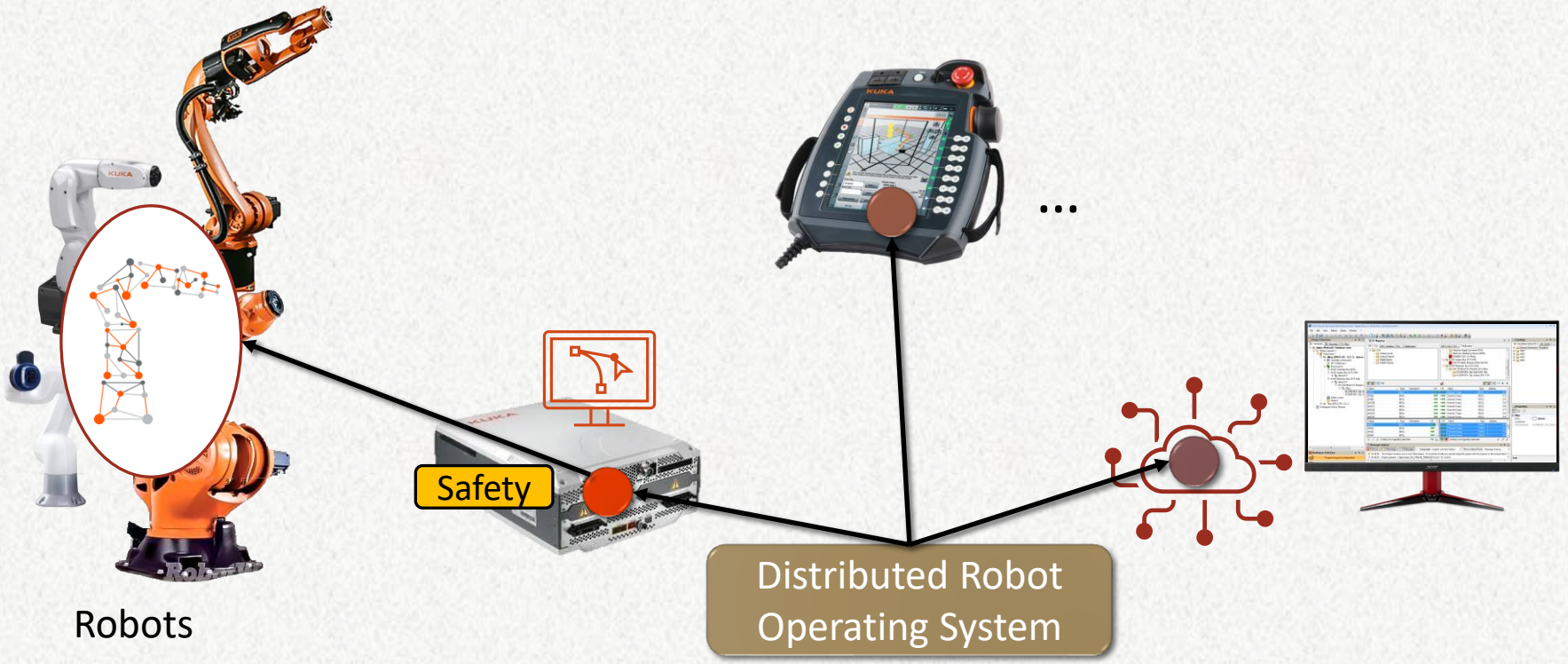
MŰEGYETEM 1782

iit

Tartalom

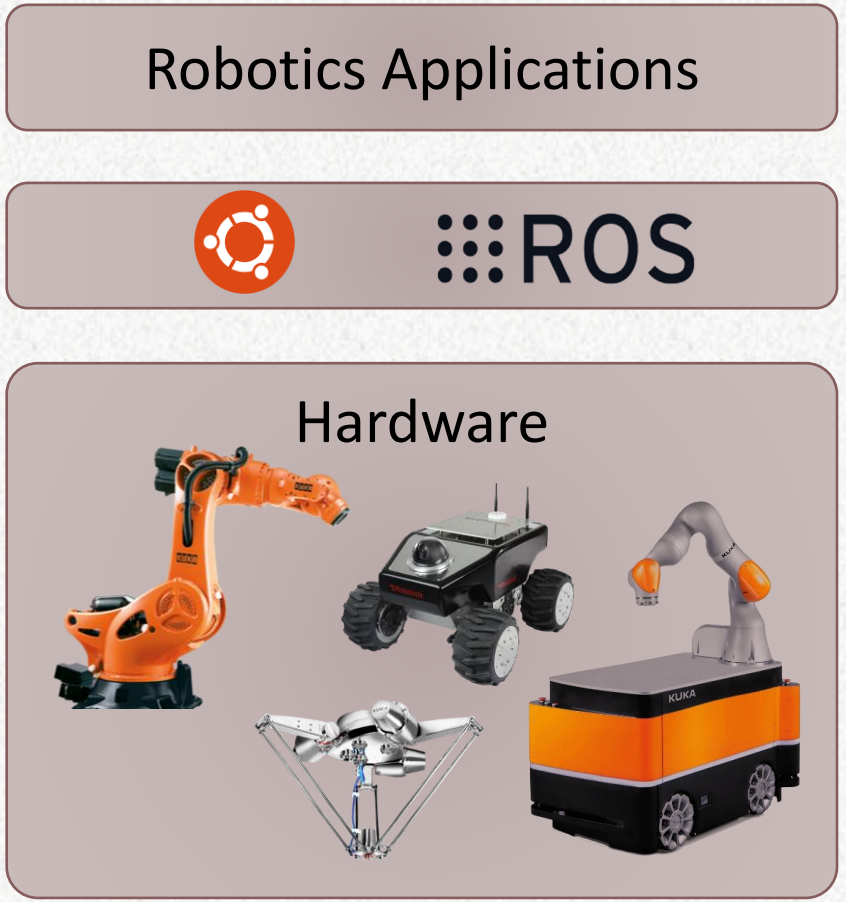
Visszatekintés

Elosztott robot irányító rendszer



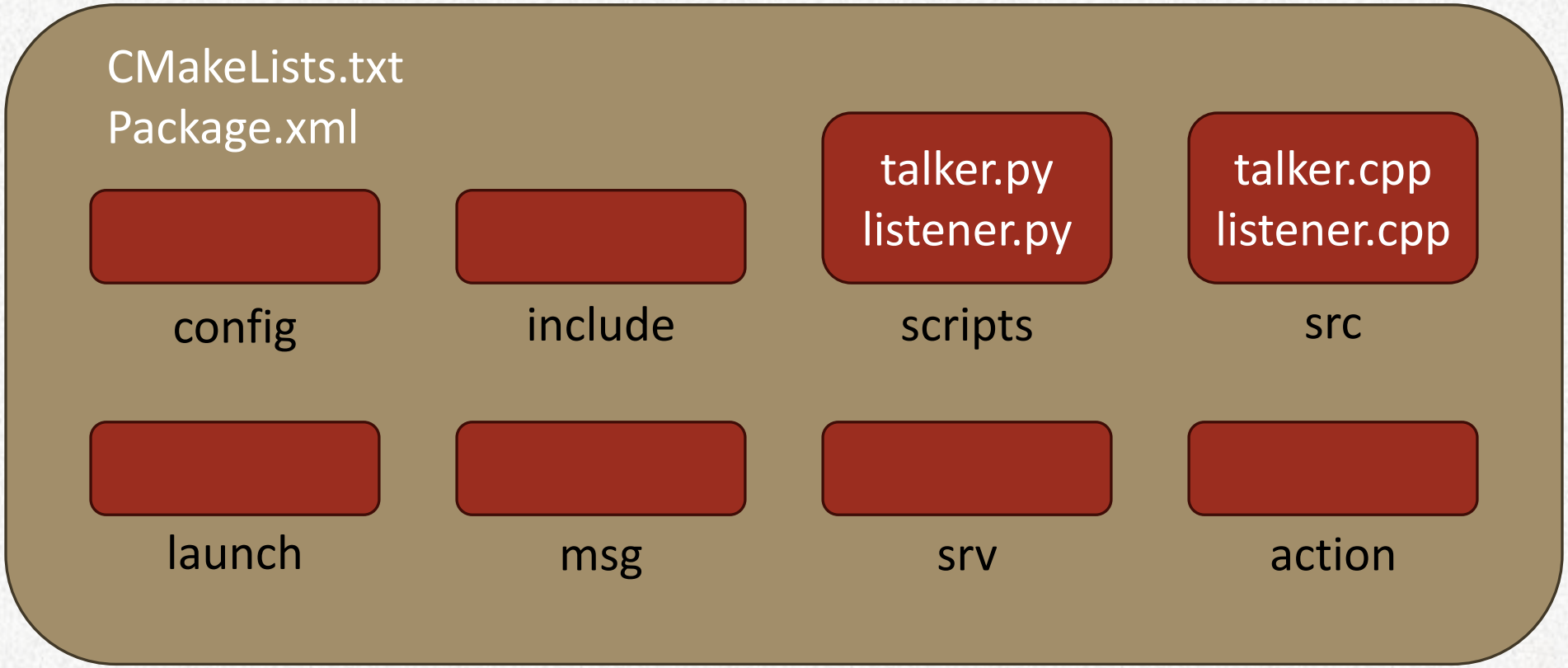
ROS – Robot Operating System

- Nem operációs rendszer
- Nyílt forráskódú middleware keretrendszer robotikai szoftverek fejlesztéséhez
- Saját könyvtár és eszközkészlet, felhasználók is létrehozhatnak ilyeneket
- Újra felhasználható kódok
- Nyelv független
- Skálázható (lazán csatolt folyamatok elosztott hálózata)
- Gráf architektúra alapú



Package – ROS alap szervezési egység

Tipikus ROS package struktúrája



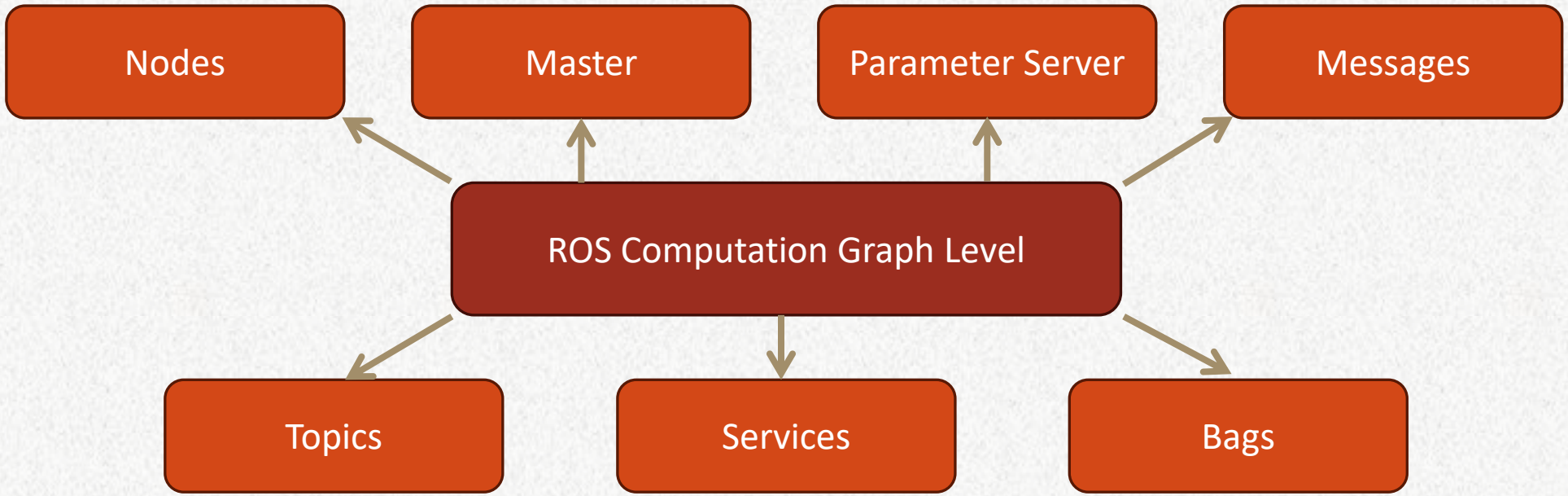
Parancssori utasítások

- `catkin_create_pkg, catkin_make`: új package létrehozása és buildelése
- `rospack, rosstack`: eszköz a package-ek, stack-ek kezeléséhez
- `roscd, rosls:` package könyvtár váltás és ROS package tartalmának listázása
- `rosdep:` ROS függőségek kezelése
- `roscp:` fájl másolás egy package-ből
- `rosed`: fájl szerkesztése
- `rosrun, roslaunch`: package futtatható állományának/állományainak futtatása
- `roscore`: roscore futtatása (ROS rendszermag: master + rosout + parameter server)
- `rosnode, rostopic, rosmsg, rossrv, rosservice`
- `rosparam`

ROS Computation graph

Computation graph

A ROS folyamatok peer-to-peer hálózatot alkotnak, melynek elemei:



ROS Node

Node

- A Node-ok műveleteket, számításokat végző folyamatok, a ROS package futtatható állományai (egyenként fordíthatók, futtathatók, menedzselhetők)
- Adott néven élnek a rendszerben
- Egyszerű folyamatok, kevés funkcionalitással
- Moduláris felépítés: egy robot szoftver tipikusan számos node-ból áll.
- ROS node-ok többféleképpen kommunikálhatnak, adatokat cserélhetnek (ld. később topic, service)
- ROS kliens könyvtárak használhatók ROS node-ok írásához (roscpp, rospy)
- Speciális node: roscore

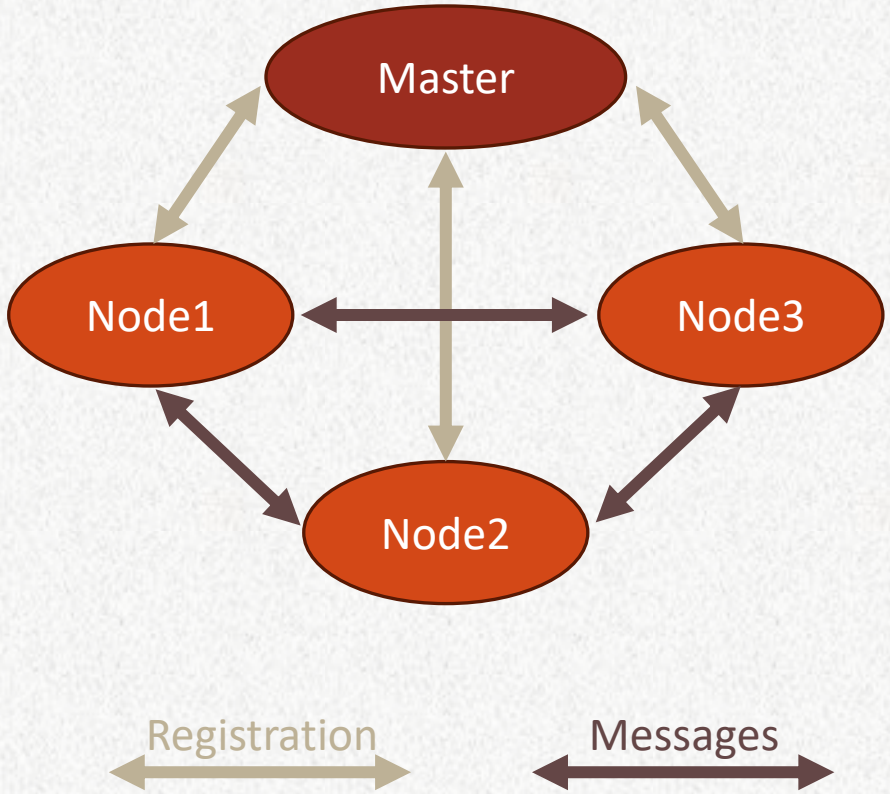
rosnode

- ROS node-okról információt kaphatunk a parancssorban

- `rosnode ping [options] <node_name>`: node elérhetősége ellenőrizhető
  `rosnode list`: aktív node-ok listázása
  `rosnode info [options] <node_name(s)>`: információkat ad meg egy node-ról
  `rosnode machine <machine_name>`: egy gépen futó node-okat listázza
  `rosnode kill <node_name>`: futó node leállítása
  `rosnode cleanup`: nem elérhető node-ok regisztrációjának törlése

Master

- ROS név szerviz szolgáltatása, név regisztráció a Computation gráf elemeihez
- A segítségével találják meg a node-ok egymást, kommunikálnak egymással
- A Master tárolja a topic-okat, service-eket, regisztrációs információt a node-ok számára
- Mindig fut a háttérben
- Egy node, amikor aktiválják, a Master-hez kapcsolódik, megadja, hogy milyen kommunikációs csatornákat használ
- A node-ok a Master-től kapják meg a szükséges információkat ahhoz, hogy kapcsolatba léphessen a többi node-dal

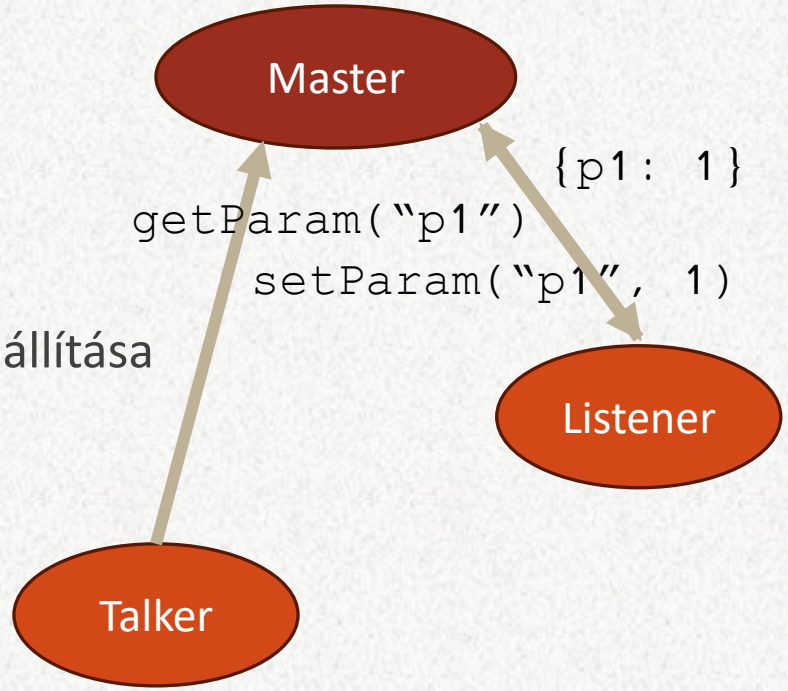


roscore

- roscore a ROS központi magja
- biztosítja az alapvető funkciókat és a node-ok kommunikációját
- Név és paraméter szerver
- Singleton (egyszerre csak egy futhat belőle)
- Futtatni kell, mielőtt bármilyen más node-ot futtatnánk
- roscore = Master + parameter server + rosout
- (rosout: ROS szabványos kimenete, a stdout/stderr megfelelője ROS-ban)

Paraméterek

- Tipikusan statikus konfigurációs paraméterek
- Aktív node-ok, ROS alkalmazások használhatják
- YAML formátum (lightweight markup language)
- Parameter server: hálózati API-n keresztül elérhető dictionary
- `rosparam set <parameter_name> [value]`: paraméter beállítása
  `rosparam get <parameter_name>`: paraméter lekérdezése
  `rosparam load <yaml_file>`: paraméterek betöltése fájlból
  `rosparam dump <yaml_file>`: paraméterek fájlba írása
  `rosparam delete <parameter_name>`: paraméter törlése
  `rosparam list`: paraméterek listázása
- launch fájlokban is használható: `<rosparam>` tag



Node-kommunikáció

ROS messages

- Node-ok message-ek segítségével kommunikálnak
- Az üzenet tartalmazza a másik node számára elküldendő információt
- Egyszerű adatstruktúra
- Sztenderd, egyszerű típusok (pl. integer, float, bool, …), struktúrák, tömbök
- Ezek felhasználásával saját message típust lehet alkotni.
- Az üzenetek típusát adattípus leírások határozzák meg, amit a ROS package `msg` nevű alkönyvtárában lévő `.msg` fájlokban adhatunk meg.

ROS message definíció

- Két típus: mező (field), konstans (constant)
- A mező estén szerepel a típusa és a neve (pl. int32 number)
- A konstans egy állandó értéket definiál az msg fájlban (pl. string pl=pelda)
- Az üzenet adatainak elérése a név alapján történik (pl. msg1.number)
- Néhány beépített mező típus →
- Message header: speciális ROS message típus, az üzenettel kapcsolatos információkhoz. Pl. üzenet sorszáma, időbélyeg, keret azonosító:

```
uint32 seq
time stamp
string frame_id
```

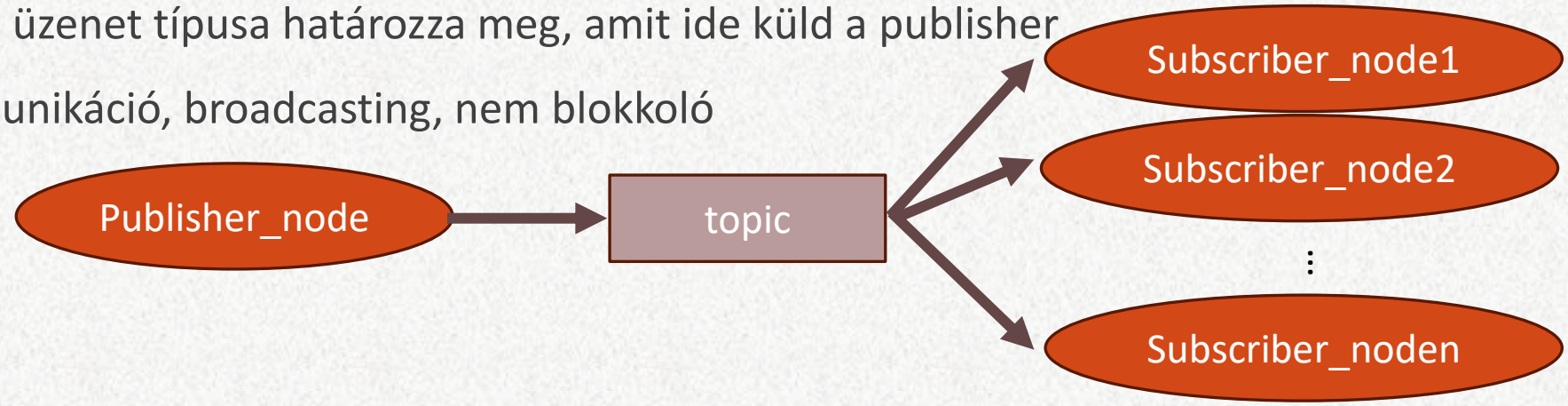
Mező típus	C++	Python
bool	uint8_t	bool
int8	int8_t	int
uint8	uint8_t	int
int16	int16_t	int
unit16	uint16_t	int
int32	int32_t	int
unit32	uint32_t	int
int64	int64_t	long
unit64	uint64_t	long
float32	float	float
float64	double	float
string	std::string	string
time	ros::Time	rospy.Time
duration	ros::Duration	rospy.Duration

rosmsg

- Információt ad a ROS üzenetek típusáról
- `rosmsg show [options] <message_type>`: Üzenet leírását adja meg
  `rosmsg info`: ld. show
  `rosmsg list`: Kilistázza az összes message-et
  `rosmsg md5 <message_type>`: md5sum üzenetet jeleníti meg
  `rosmsg package <package_name>`: Egy package üzeneteit listázza
  `rosmsg packages`: Package-eket listáz, amik tartalmaznak üzeneteket
- md5sum: egy hash, a ROS message definition fájlból számítják. Ellenőrizhető vele, hogy egy üzenet küldője és vevője ugyanazzal a definícióval dolgozik-e.

Topics – publish/subscribe

- Az üzenetküldésre egy lehetőség a topic-ok használata
- Publish – egy node üzenetet küld egy topic-ra
- Subscribe – egy node üzenetet szeretne kapni egy topic-on keresztül
- A topic szétcsatolja az üzenet előállítását és a felhasználását
- Minden topic-nak egyedi neve van
- Típusát az üzenet típusa határozza meg, amit ide küld a publisher
- 1:n kommunikáció, broadcasting, nem blokkoló



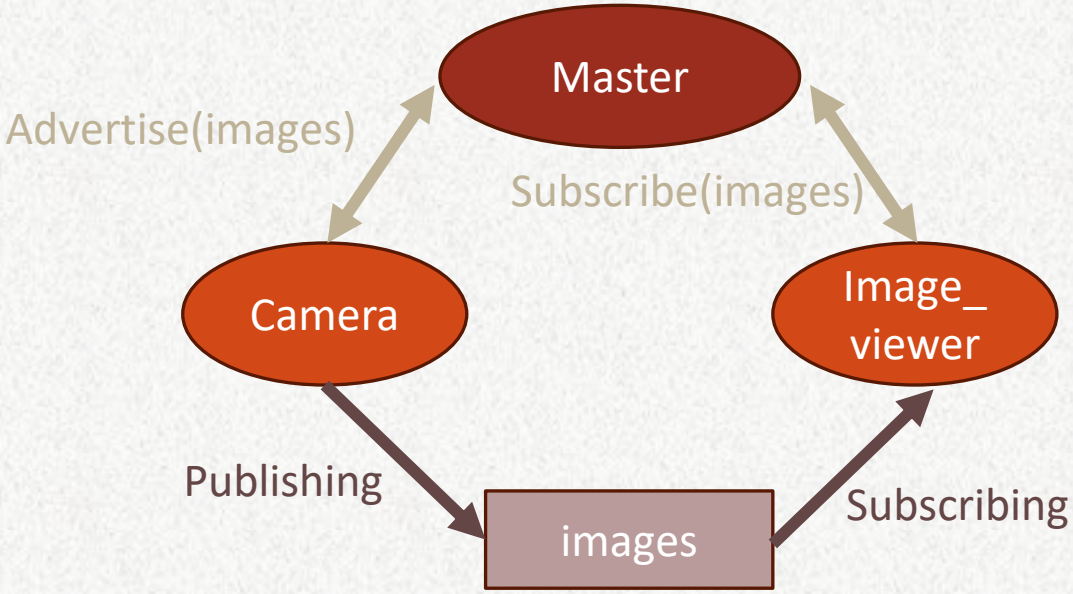
rostopic

- Parancssori utasítások, melyekkel információt kaphatunk a ROS topic-okról

- `rostopic bw <topic_name>`: topic által használt sávszélességet írja ki
  `rostopic delay [options] <topic_name>`: késleltetés (headerben lévő időbélyeghez képest)
  `rostopic echo [options] <topic_name>`: topic üzeneteit megjeleníti a képernyőn
  `rostopic find <message_type>`: üzenet típus alapján keres topic-ot
  `rostopic hz [options] <topic_name(s)>`: topic üzenetküldési gyakoriságát írja ki
  `rostopic info <topic_name>`: aktív topic-ról ad információt
  `rostopic list [namespace]`: aktív topic-ok listázása
  `rostopic pub <topic> <type> [args]`: adatokat publikál egy topic-on
  `rostopic type <topic_or_field_name>`: topic vagy mező típusát adja meg

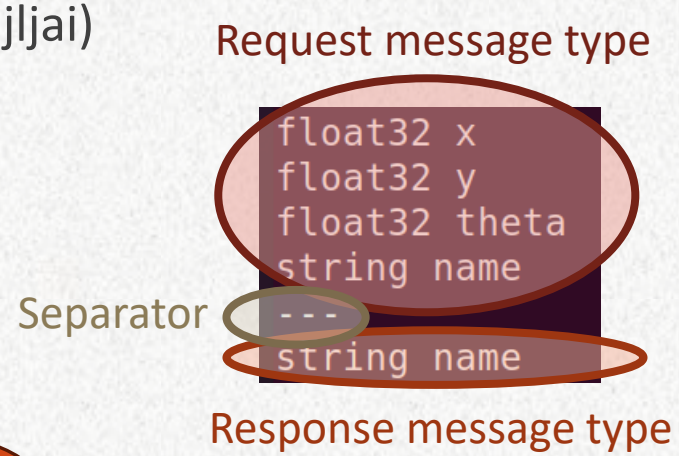
Példa

- Két node: Camera, Image_viewer
- Camera jelzi a Master-nak, hogy az images topic-ra szeretne üzeneteket küldeni
- Az Image_viewer jelzi a Master-nek, hogy olvasni szeretné az images topic-ot
- A Master-től kapott információk alapján az Image_viewer hozzáfér az images topic-hoz (subscriber), amire a Camera node küldi az üzeneteket (publisher)



Services – request/response

- Kérés-válasz típusú kommunikáció
- Server node – egy adott névvel service-t biztosít
- Kliens node – request üzenetet küld, ha szüksége van valamire, majd válaszra vár
- A service-ek is használnak ROS message definíciókat (srv könyvtár .srv fájljai)
- Üzenet struktúra pár: egy a request-nek, egy a response-nak
- 1:1 kommunikáció, blokkoló



rossrv

- Információt kaphatunk a ROS service típusokról
- `rosmsg`-hez hasonló utasítások
- `rossrv show [options] <service_type>`: service leírását adja meg
  `rossrv info`: ld. `rossrv`
  `rossrv list`: minden service-t kilistáz
  `rossrv md5 <service_type>`: md5sum hash-t adja meg
  `rossrv package <package_name>`: egy package service-eit listázza
  `rossrv packages`: megadja a package-eket, mik tartalmaznak service-eket

rosservice

- ROS service-ek listázása és lekérdezése végezhető parancssorból

- `rosservice args <service_name>`: service argumentumainak listázása
  `rosservice call <service_name> [args]`: service meghívása a megadott argumentumokkal
  `rosservice find <message_type>`: service típus alapján keres service-ket
  `rosservice info <service_name>`: információt ír ki egy service-ről
  `rosservice list [namespace]`: aktív service-eket listázza
  `rosservice type <service_name>`: service típusát adja meg
  `rosservice uri <service_name>`: kiírja a service ROSRPC uri címét

Példa node-kommunikációra

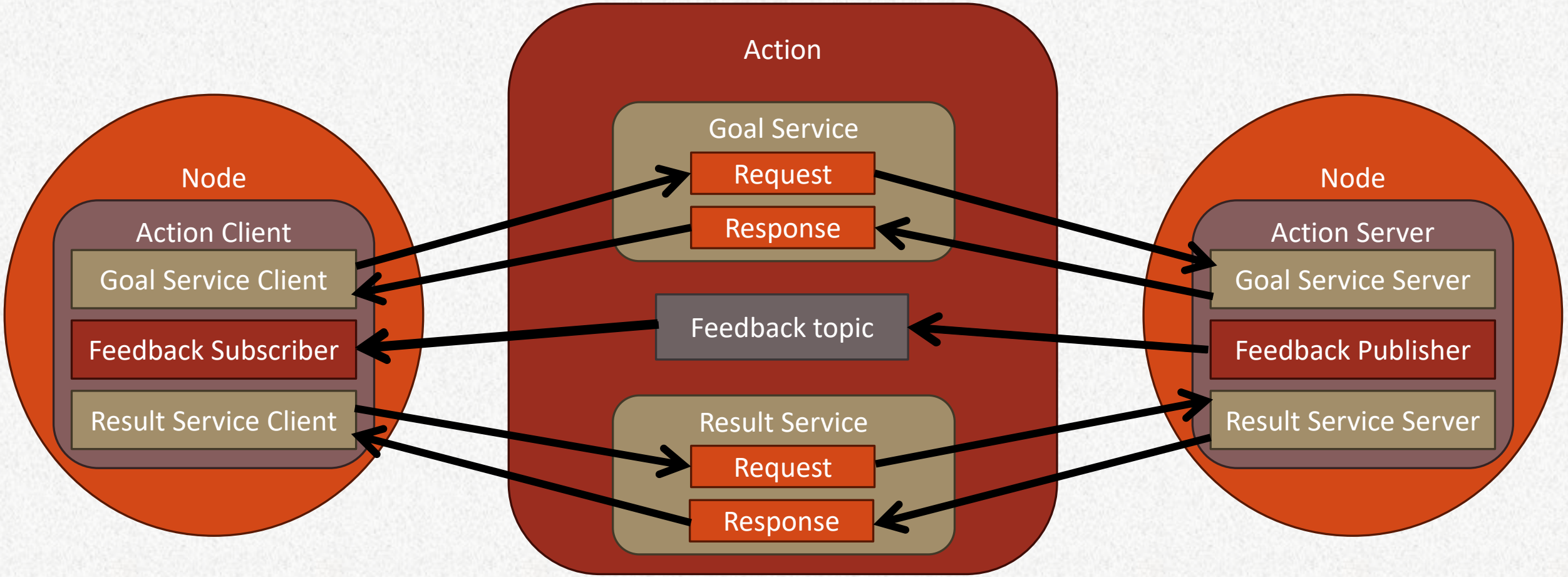
```
emese@ubuntu:~/emese/ros$ rosnode info turtlesim
--------------------------------------------------------------------------------
Node [/turtlesim]
Publications:
 * /rosout [rosgraph_msgs/Log]
 * /turtle1/color_sensor [turtlesim/Color]
 * /turtle1/pose [turtlesim/Pose]

Subscriptions:
 * /turtle1/cmd_vel [geometry_msgs/Twist]

Services:
 * /clear
 * /kill
 * /reset
 * /spawn
 * /turtle1/set_pen
 * /turtle1/teleport_absolute
 * /turtle1/teleport_relative
 * /turtlesim/get_loggers
 * /turtlesim/set_logger_level
```

```
contacting node http://ubuntu:40785/ ...
Pid: 20398
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound (58233 - 127.0.0.1:38642) [27]
    * transport: TCPROS
 * topic: /turtle1/cmd_vel
    * to: /teleop_turtle (http://ubuntu:37345/)
    * direction: inbound (51886 - ubuntu:49861) [25]
    * transport: TCPROS
```

Action

- actionlib stack
- service jellegű request/response, de preemptív
- Akkor használatos, ha hosszabb időre van szükség a válasz küldéséhez
- Egy node-dal egy műveletet elvégeztet (goal), ami végén választ vár (result)
- A kérés visszavonható, vagy periodikus értesítés kérhető (feedback), amíg tart a művelet
- Definíció: .action fájlokban, amik három részből állnak:
  - goal
  - result
  - feedback message
  - elválasztás: ---

Action

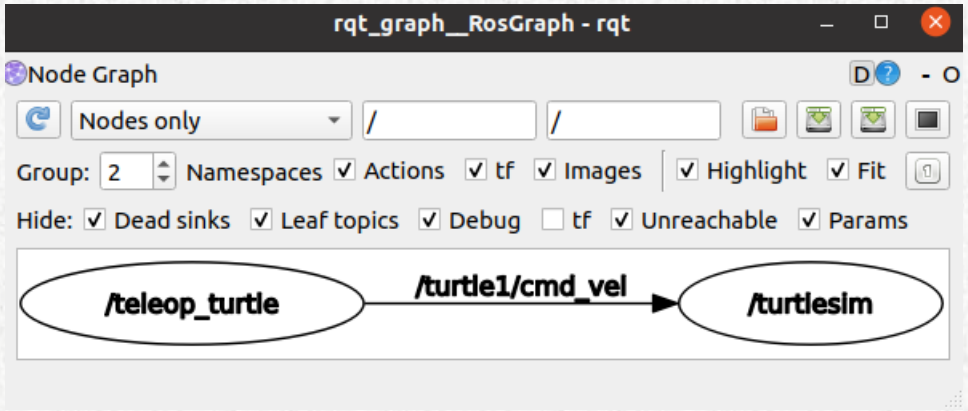

Action
Goal Service
Request
Response
Feedback topic
Result Service
Request
Response
Node
Action Client
Goal Service Client
Feedback Subscriber
Result Service Client
Node
Action Server
Goal Service Server
Feedback Publisher
Result Service Server

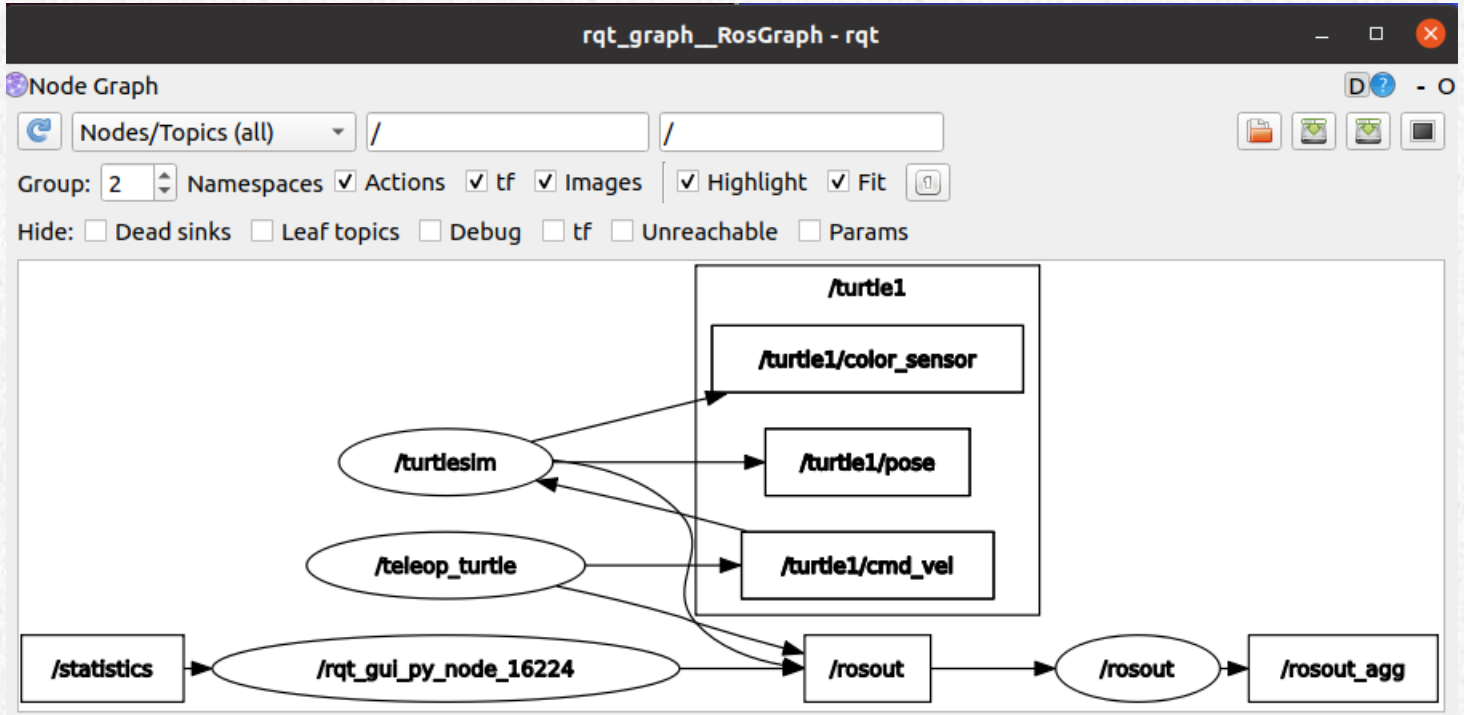
További ROS eszközök

Gráf megjelenítés

- rqt_graph – ROS computation gráf vizuális megjelenítését végző plugin
- Megjeleníti az aktív node-okat (ellipszisek) és a köztük zajló kommunikációt (nyíl vagy téglalap)
- Futtatás:

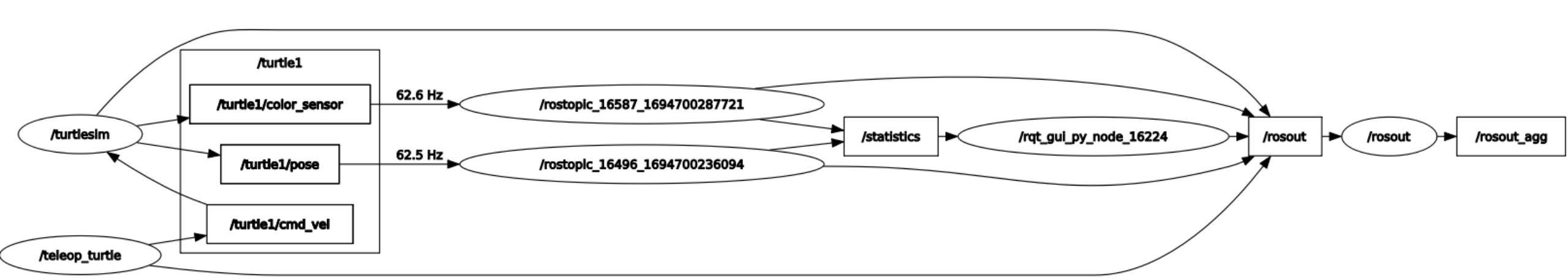
```
rosrun rqt_graph rqt_graph
```





Gráf megjelenítés

- Topic-okhoz tartozó statisztikai adatok is megjeleníthetők (pl. frekvencia)
- `$ rosparam set enable_statistics true` # rqt_graph futtatása előtt
  `$ rosrun rqt_graph rqt_graph`



Rosbag

- Adat loggolásra használható
- A bag-ek feliratkoznak egy vagy több ROS topic-ra, és elmentik a szerializált üzenet adatokat egy fájlba
- A bag fájlok visszajátszhatók a ROS-ban ugyanarra a topic-ra, ahonnan lementették az adatokat
- Új topic-on keresztül is adhatják az adatokat
- Tipikus alkalmazás: mérés elvégzése valós szenzorral, adatok tárolása rosbag-ben, algoritmus fejlesztés és tesztelés céljából a mentett adatok használata (valós szenzor nélkül)

rosbag

- rosbag utasítással bag-be mentés, visszajátszás és módosítás is lehetséges
- `rosbag check`: annak ellenőrzése, hogy egy bag fájl az aktuális rendszerben lejátszható-e (vagy migrálható-e)

`rosbag compress`: egy vagy több bag fájl tömörítése
`rosbag decompress`: egy vagy több bag fájl kitömörítése
`rosbag decrypt`: egy vagy több titkosított bag fájl
`rosbag encrypt`: egy vagy több bag fájl titkosítása
`rosbag filter`: bag fájl tartalmának szűrése
`rosbag fix`: bag fájl üzeneteinek javítása (az aktuális rendszernek megfelelően)
`rosbag info`: bag fájlok tartalmáról rövid információ
`rosbag play`: egy vagy több bag fájl tartalmának visszajátszása
`rosbag record`: topic mentése bag fájlba
`rosbag reindex`: egy vagy több bag fájl átindexelése

Nodelet

- Node-ok közti kommunikáció sok adat küldéséhez vezethet (pl. kamera képek küldése topic-okon keresztül)
- Erre ad megoldást a nodelet package
- Egy folyamatban futhatnak különböző algoritmusok
- Zero-copy
- Van egy nodelet manager, amibe betölthetők a nodeletek.
- Részletek: https://wiki.ros.org/nodelet

Programozás

Catkin workspace létrehozása

```
emese@ubuntu:~$ mkdir ~/catkin_ws # create folder for catkin workspace
emese@ubuntu:~$ cd catkin_ws
emese@ubuntu:~/catkin_ws$ mkdir src # create folder for source space
emese@ubuntu:~/catkin_ws$ catkin_make # build workspace
Base path: /home/emese/catkin_ws
Source space: /home/emese/catkin_ws/src
Build space: /home/emese/catkin_ws/build
Devel space: /home/emese/catkin_ws/devel
Install space: /home/emese/catkin_ws/install
```

```
emese@ubuntu:~/catkin_ws$ echo $ROS_PACKAGE_PATH # check package path
/opt/ros/noetic/share
emese@ubuntu:~/catkin_ws$ source ~/catkin_ws/devel/setup.bash # add new workspace
emese@ubuntu:~/catkin_ws$ echo $ROS_PACKAGE_PATH # check package path again
/home/emese/catkin_ws/src:/opt/ros/noetic/share
```

Package létrehozása

```
emese@ubuntu:~/catkin_ws$ cd src
emese@ubuntu:~/catkin_ws/src$ catkin_create_pkg my_pkg # create new package
Created file my_pkg/package.xml
Created file my_pkg/CMakeLists.txt
Successfully created files in /home/emese/catkin_ws/src/my_pkg. Please adjust the value
s in package.xml.
```

```
emese@ubuntu:~/catkin_ws/src$ ls # check the src folder
CMakeLists.txt  my_pkg
emese@ubuntu:~/catkin_ws/src$ ls my_pkg # check the my_pkg folder
CMakeLists.txt  package.xml
```

```
emese@ubuntu:~/catkin_ws/src$ cd ..
emese@ubuntu:~/catkin_ws$ catkin_make # build the package
Base path: /home/emese/catkin_ws
Source space: /home/emese/catkin_ws/src
Build space: /home/emese/catkin_ws/build
Devel space: /home/emese/catkin_ws/devel
Install space: /home/emese/catkin_ws/install
```

Msg, srv, talker - listener

- msg fájl létrehozása
- talker.cpp, listener.cpp forrásfájlok megírása
  - vagy talker.py, listener.py node-ok pythonban
- srv fájl létrehozása (további infó)
- szerver-kliens implementálása C++-ban
  - vagy Pythonban
- projekt újrafordítása: `catkin_make` (workspace könyvtárában)
- roscore futtatása: `roscore`
- node-ok futtatása: `rosrun`