

# Robot operációs rendszerek és fejlesztői ökoszisztemák

## ROS alapok 1.

---

Gincsainé Szádeczky-Kardoss Emese

2025. szeptember 15.

**KUKA**



**iit**

# Tartalom

---

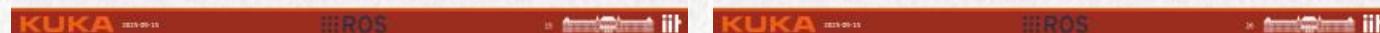
Visszatekintés

Mi is az a ROS?



ROS Fájlrendszer szint

rosbash parancsok



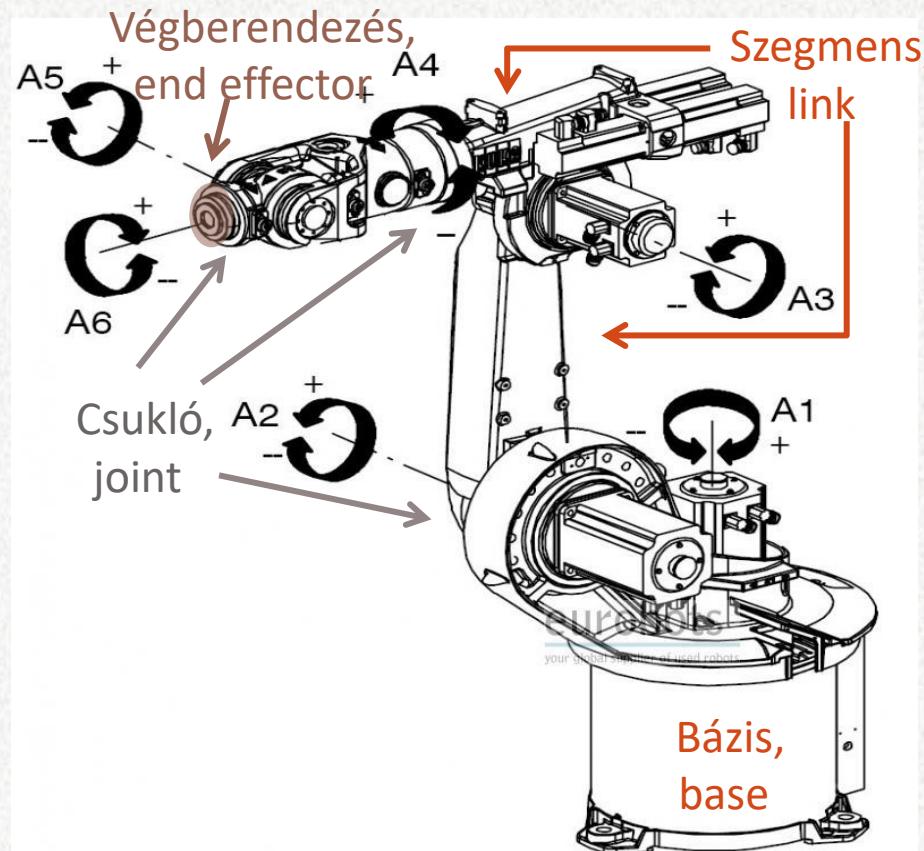
# Visszatekintés

---

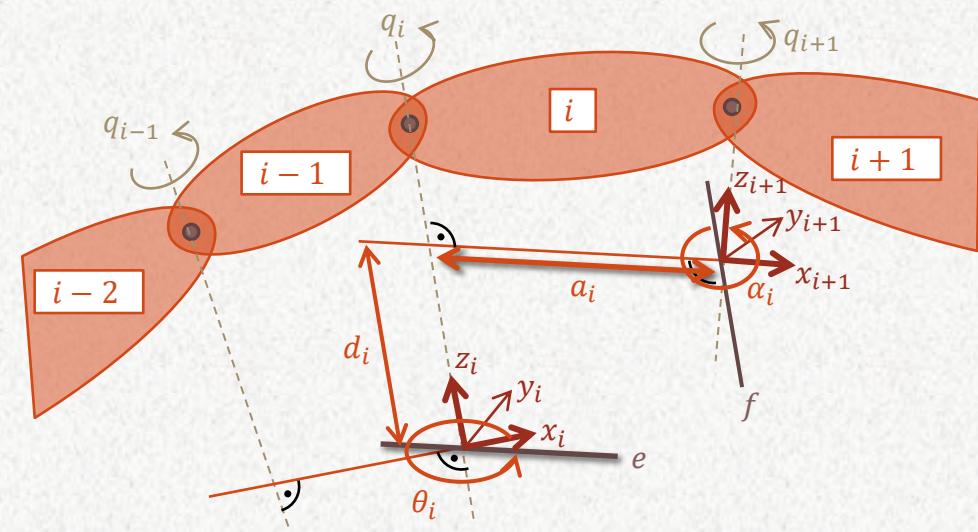
# Robotikai alapok



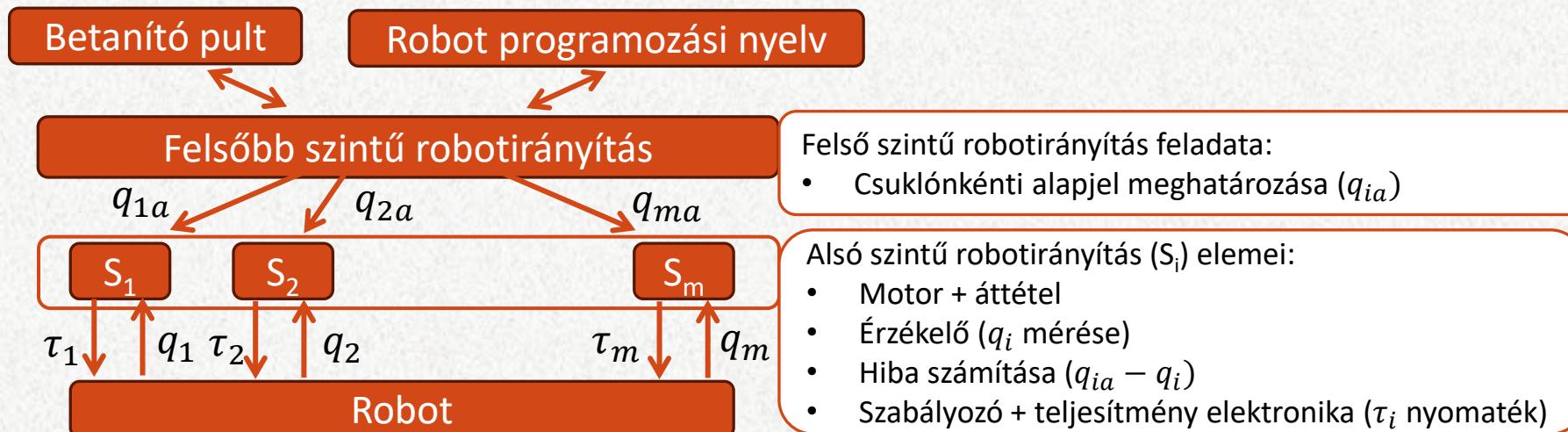
# Robotikai alapok



Robotkar geometriai modellezése DH paraméterekkel

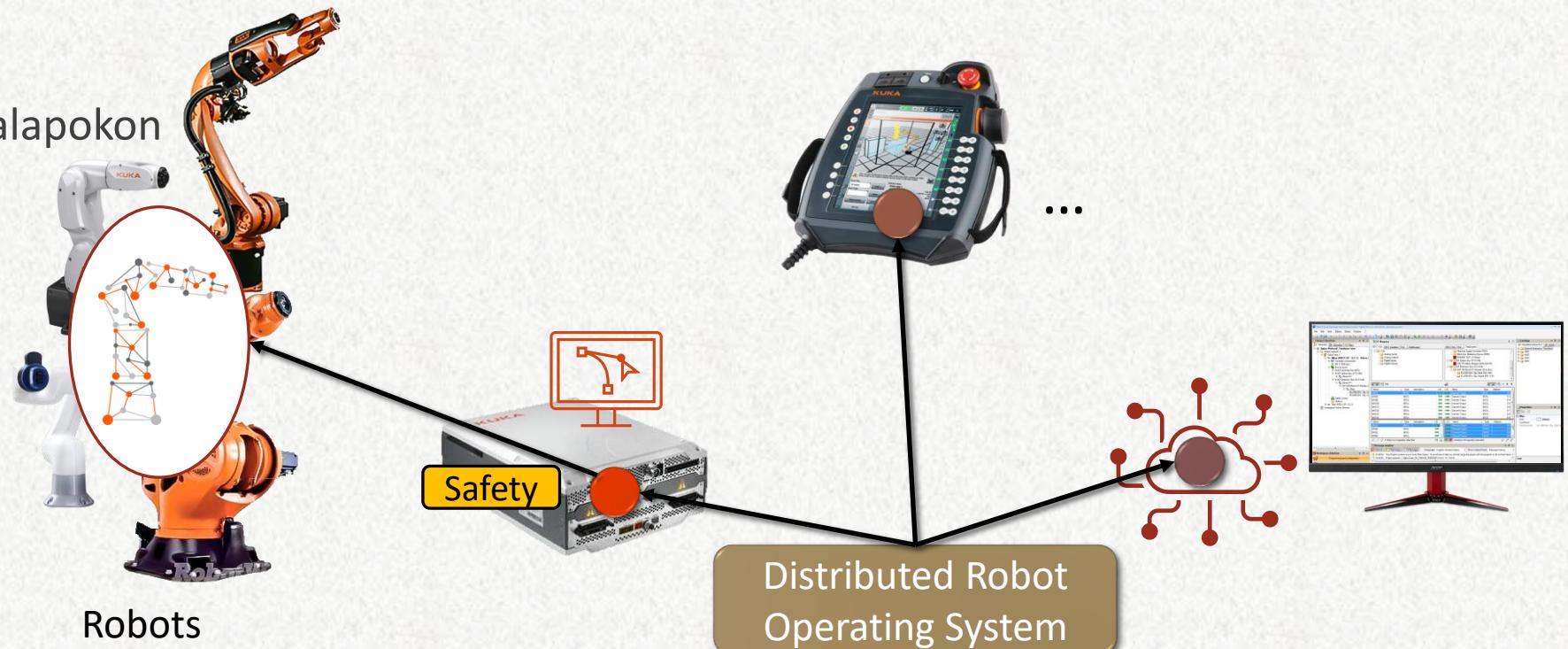


# Robotikai alapok



# SoA rendszerek

- Service-oriented Architecture
- Célok áttekintése
- Service – microservice
- Robotrendszerök SoA alapokon
- Nehézségek
- Ideális robot OS



# Mi is az a ROS?

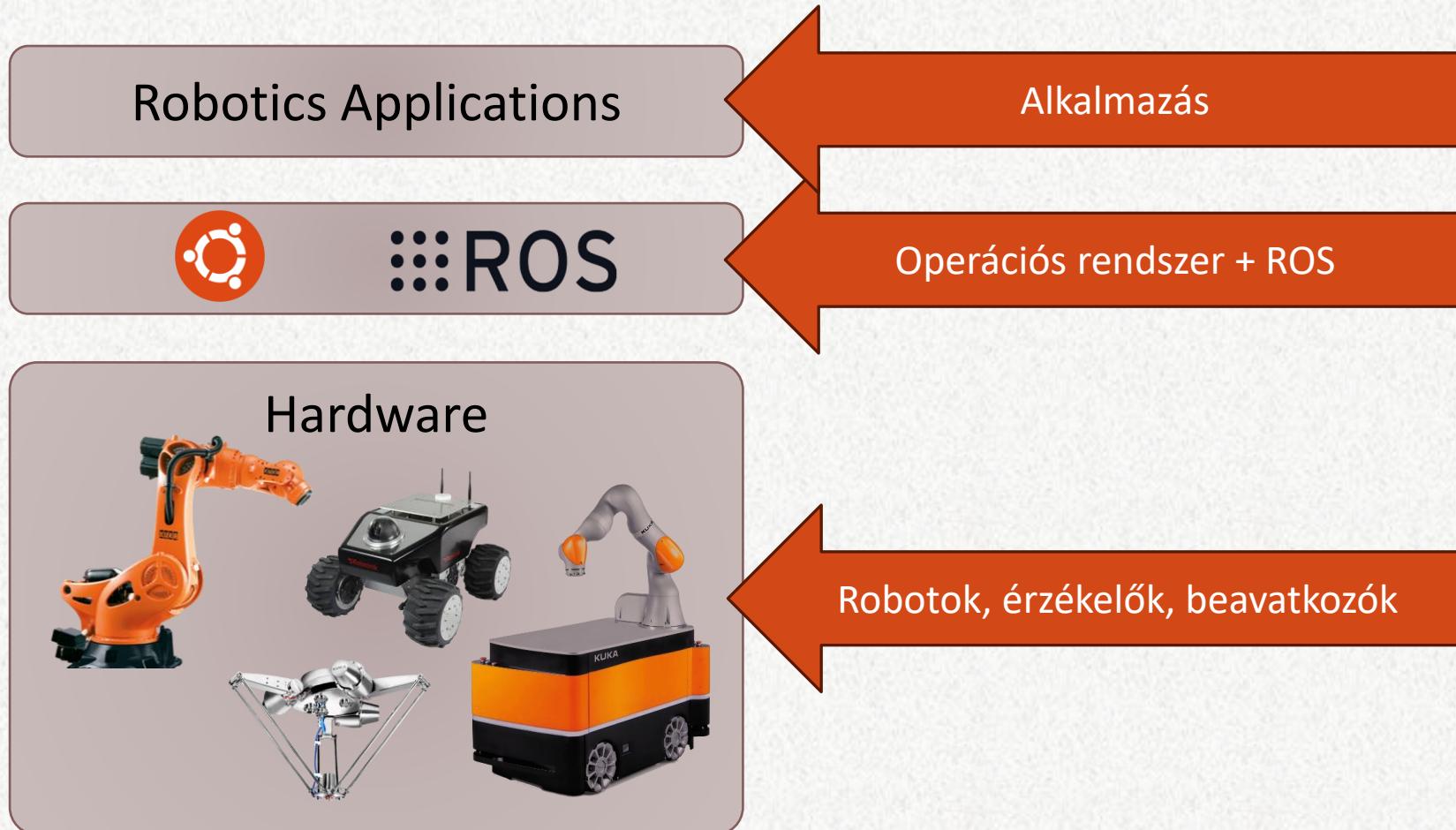
---

# ROS – Robot Operating System

---

- Nem operációs rendszer
- Nyílt forráskódú flexibilis szoftver keretrendszer
- Elsősorban robotikai alkalmazások fejlesztéséhez használható
- Vannak operációs rendszer jellegű szolgáltatásai (hardver absztrakció, alacsony szintű eszközkezelés, folyamatok közötti üzenetküldés, package management).
- Saját könyvtár és eszközkészlet, felhasználók is létrehozhatnak ilyeneket
- Újra felhasználható kódok
- Célja, hogy egyszerűsítse a fejlesztést különböző heterogén (robot)platformok esetén
- BSD (Berkeley Software Distribution) licences

# ROS – Robot Operating System



# ROS tulajdonságok

---

- Újra felhasználható kódok
- Használatra kész fejlesztői környezet, sokféle eszköz, kliens API könyvtárak
- Nyelv független
- Skálázható (lazán csatolt folyamatok elosztott hálózta)
- Gráf architektúra alapú, gráfpontok: node-ok, egyszerű folyamatok
- Folyamatok közötti kommunikáció biztosítása
- Folyamatok (node-ok) package-ekbe, stack-ekbe csoportosíthatók
- Közös robot-specifikus könyvtárak, eszközök (pl. robot leíró nyelv)
- Hibakeresést, vizualizációt támogató eszközök
- Nagy közösség, folyamatos support

# ROS történet

---

- 2000-es évek közepén STAIR (Stanford AI Robot), Personal Robot (PR) projektek – szoftver rendszer kidolgozása robotikai célokra
- 2007. Willow Garage – robotikával foglalkozó inkubátorház
- 2010. első ROS disztribúció
- Jelenleg Open Source Robotics Foundation (<https://www.openrobotics.org>)
- 2014-ben indult a ROS2 fejlesztése
- 2017. első ROS2 disztribúció

# ROS distros

---

Első Java  
release?

Vajon milyen nap lehet  
május 23?

Dezső?



Teknősök  
világnapja?

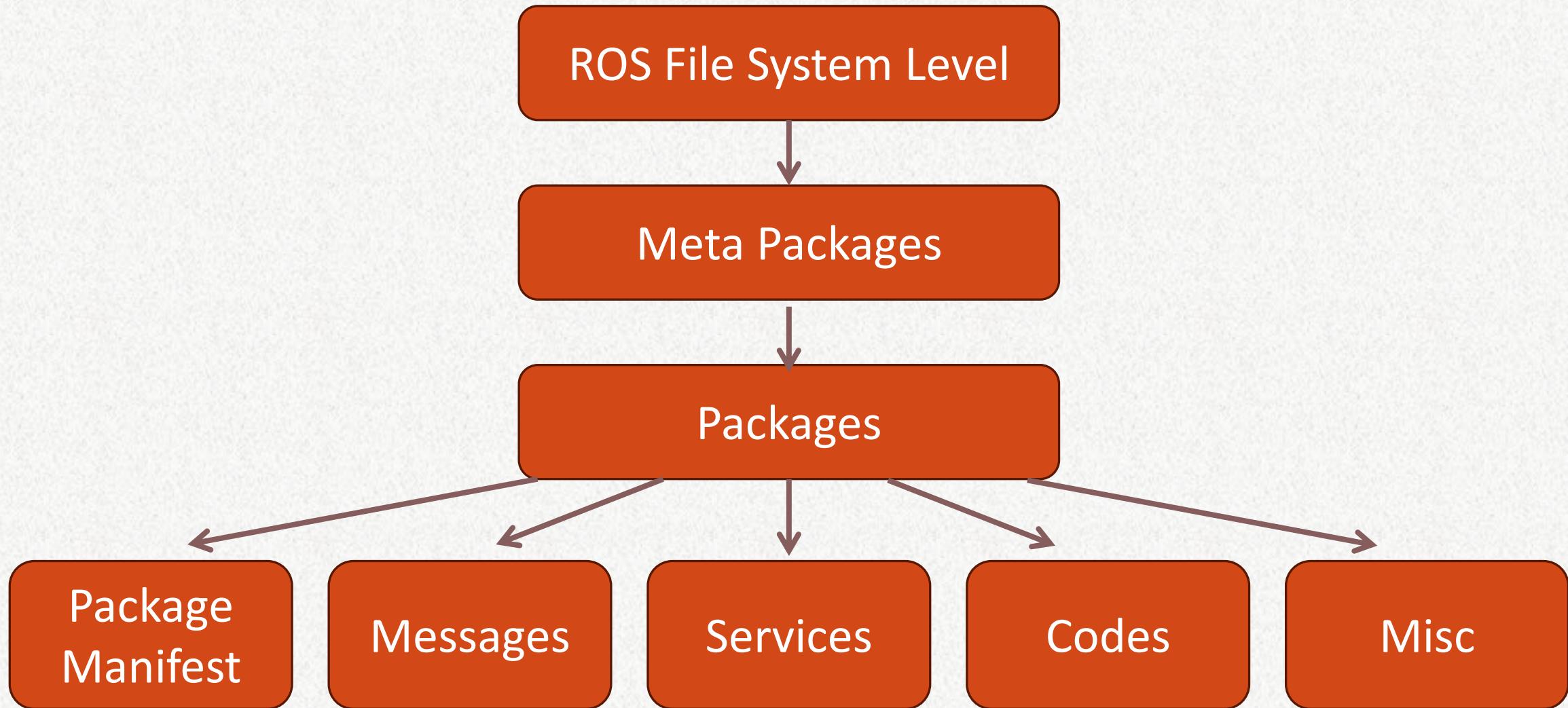
# ROS distros

---

- ROS disztribúciók
- Legfrissebbeket rendszeresen május 23-án adták ki (teknősök világnapja)
- ROS1: <http://wiki.ros.org/Distributions>
- ROS2: <https://docs.ros.org/en/rolling/Releases.html>
- Telepítéshez az információk elérhetők az adott disztribúciók oldalán.
- Telepítés után minden egyes terminálban (Environment setup):  
`$ source /opt/ros/<distro>/setup.bash`  
vagy egyszerűbben  
`$ echo "source /opt/ros/<distro>/setup.bash" >> ~/.bashrc`  
`$ source ~/.bashrc`

# ROS Fájlrendszer szint

---



# ROS File System Level

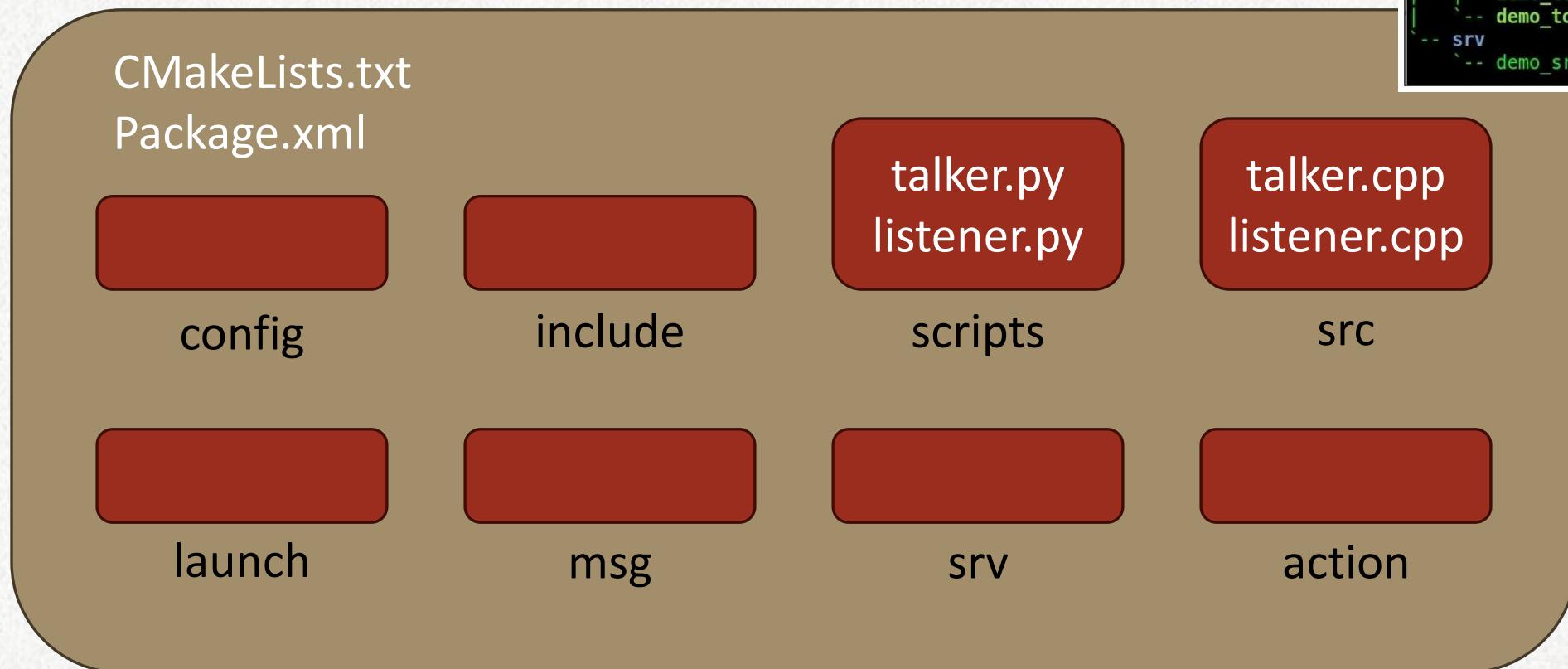
---

## Lemezen tárolt fájlok, könyvtárak

- **Package** (csomag): ROS szoftver alap szervezési egysége. Node-okat, könyvtárakat, dataset-eket, konfigurációs fájlokat, dokumentációt tartalmaz
- **Package Manifest**: ROS csomag leíró, package-hez tartozó metaadatokról szolgáltat információt (pl. szerző, licensz, függőségek) – `manifest.xml`, `package.xml`
- **Meta package / Stack**: Package-ek gyűjteménye, amik egy speciális, összetett funkcióhoz tartoznak (pl. navigation stack)
- **Meta package / Stack Manifest**: Stack-hez tartozó adatokat tartalmazza (pl. függések más stack-től) – `stack.xml`, `package.xml`

# Package

Tipikus ROS package struktúrája



```
mastering_ros_demo_pkg/
|-- action
|   '-- Demo_action.action
-- CMakeLists.txt
-- include
-- msg
|   '-- demo_msg.msg
-- package.xml
-- src
|   '-- demo_action_client.cpp
|   '-- demo_action_server.cpp
|   '-- demo_msg_publisher.cpp
|   '-- demo_msg_subscriber.cpp
|   '-- demo_service_client.cpp
|   '-- demo_service_server.cpp
|   '-- demo_topic_publisher.cpp
|   '-- demo_topic_subscriber.cpp
-- srv
`-- demo_srv.srv
```

# Package

---

- **config**: package-hez tartozó konfigurációs fájlok (Felhasználó hozza létre.)
- **include**: headers, libraries, amikre a package-en belül szükség van
- **scripts**: futtatható szkriptek
- **src**: forrás fájlok (node-okat implementálnak)
- **launch**: launch fájlok egy vagy több ROS node indításához
- **msg**: Üzenet típusok definiálása egyszerű szöveges fájlokban. Kiterjesztés: .msg
- **srv**: Definíciók service-ekhez. Két rész: request és response adattípusok. Kiterjesztés: .srv
- **action**: Definíció action-ökhöz.
- **CMakeLists.txt**: CMake build file
- **package.xml**: manifest, csomagleíró

# Package, package manifest

- A package egy könyvtár, amiben – többek között – megtalálható a hozzá tartozó manifest fájl.
- <name> - a csomag neve
- <version> - a csomag verziószáma
- <description> - leírás a csomagról, tartalmáról
- <maintainer> - szerző neve
- <license> - pl. BSD, GPL, ASL
- <build depend> - fordításhoz szükséges csomagok
- <run depend> - a package node-jainak futtatása során szükséges csomagok

```
1 <?xml version="1.0"?>
2 <package>
3   <name>turtlesim</name>
4   <version>0.10.2</version>
5   <description>
6     turtlesim is a tool made for teaching ROS and ROS packages.
7   </description>
8   <maintainer email="dthomas@osrfoundation.org">Dirk Thomas</maintainer>
9   <license>BSD</license>
10
11  <url type="website">http://www.ros.org/wiki/turtlesim</url>
12  <url type="bugtracker">https://github.com/ros/ros\_tutorials/issues</url>
13  <url type="repository">https://github.com/ros/ros\_tutorials</url>
14  <author>Josh Faust</author>
15
16  <buildtool_depend>catkin</buildtool_depend>
17
18  <build depend>geometry_msgs</build depend>
19
20  <run depend>std_srvs</run depend>
21
22  </package>
```

# Package létrehozása

---

- Egyszerű package létrehozása:

- Egy könyvtár (workspace) és egy alkönyvtár létrehozása (src)
- A package könyvtár létrehozható template alapján a source könyvtárban
- Vagy a `catkin_create_pkg` parancs használatával

`catkin_create_pkg <package_name> [depend1] [depend2] [depend3]`  
pl.

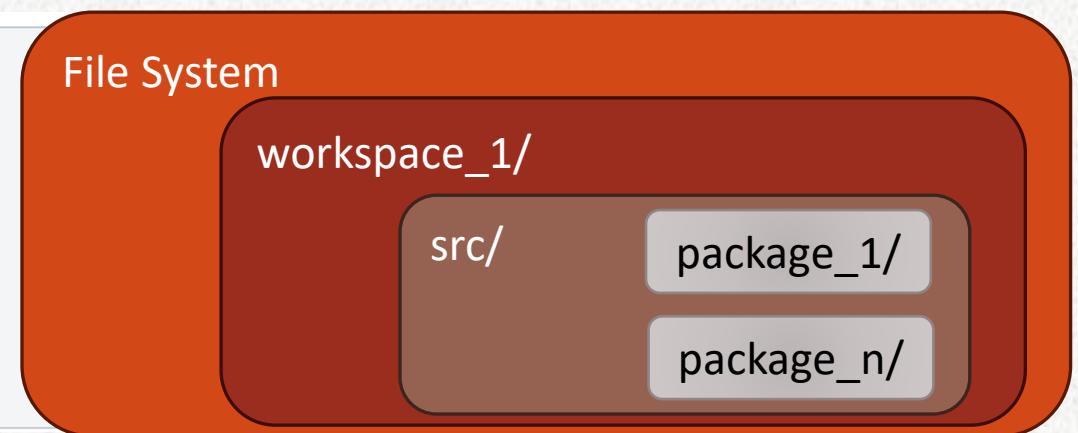
```
emese@ubuntu:~/ws1/src$ catkin_create_pkg my_pkg std_msgs
Created file my_pkg/package.xml
Created file my_pkg/CMakeLists.txt
Successfully created files in /home/emese/ws1/src/my_pkg. Please
adjust the values in package.xml.
emese@ubuntu:~/ws1/src$ catkin_create_pkg my_pkg2 std_msgs roscpp
Created file my_pkg2/package.xml
Created file my_pkg2/CMakeLists.txt
Created folder my_pkg2/include/my_pkg2
Created folder my_pkg2/src
Successfully created files in /home/emese/ws1/src/my_pkg2. Please
adjust the values in package.xml.
```

- Package build: `catkin_make` utasítással

# Catkin

- ROS build system
- Catkin workspace: ezen belül vannak a package könyvtárak

```
workspace_folder/      -- WORKSPACE
  src/                 -- SOURCE SPACE
    CMakeLists.txt     -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt   -- CMakeLists.txt file for package_1
      package.xml      -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt   -- CMakeLists.txt file for package_n
      package.xml      -- Package manifest for package_n
```



- Létezhet több workspace, de egyszerre csak egy használható
- [Workspace létrehozása](#)
- [Building catkin packages](#)

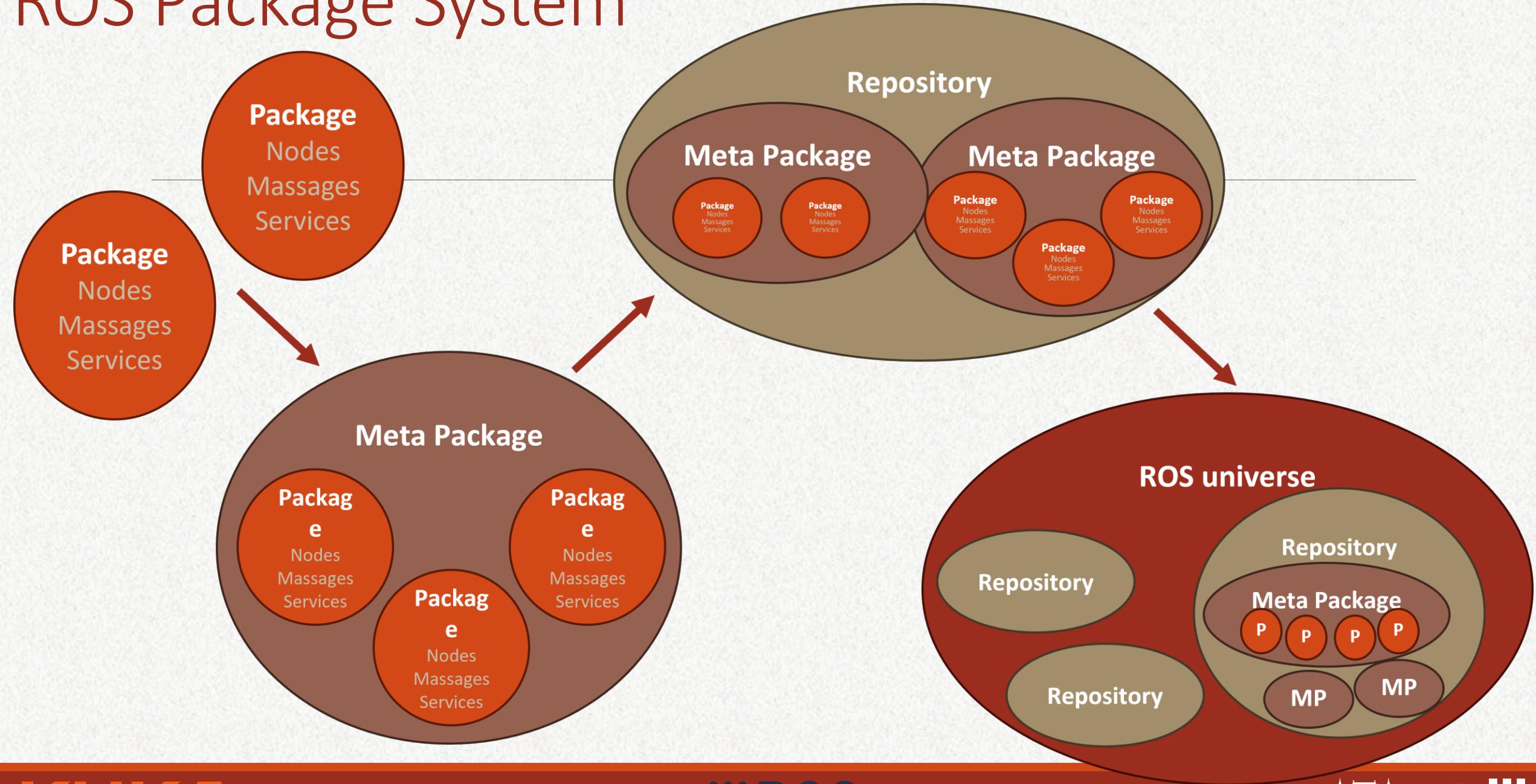
# Meta packages (stacks)

---

- A meta package több package-et összefog egy logikai package-be.
- Speciális ROS package, amiben egy `package.xml` fájl van.
- Nincsenek a normál package-hez hasonló további fájlok vagy könyvtárak.
- A `package.xml` fájl tartalmaz még egy tag-ot: `<export>`
- Nincsenek `<build_depend>` tag-ek, csak `<run_depend>`, amik azokat a package-eket tartalmazzák, amiket a meta package összefog.

```
<package>
  <name>navigation</name>
  <version>1.12.2</version>
  .....
  <buildtool_depend>catkin</buildtool_depend>
  .....
  <run_depend>amcl</run_depend>
  <run_depend>carrot_planner</run_depend>
  .....
  <export>
    <metapackage/>
  </export>
</package>
```

# ROS Package System



# ROS környezeti változók

## Környezeti változók lekérdezése

```
emese@ubuntu:/opt/ros/noetic/share$ export | grep ROS
declare -x ROSLISP_PACKAGE_DIRECTORIES=""
declare -x ROS_DISTRO="noetic"
declare -x ROS_ETC_DIR="/opt/ros/noetic/etc/ros"
declare -x ROS_MASTER_URI="http://localhost:11311"
declare -x ROS_PACKAGE_PATH="/opt/ros/noetic/share"
declare -x ROS_PYTHON_VERSION="3"
declare -x ROS_ROOT="/opt/ros/noetic/share/ros"
declare -x ROS_VERSION="1"
```

## Környezeti változók módosítása

Pl. `export ROS_PACKAGE_PATH="/home/emese/ros/catkin_ws"`

# rosbash parancsok

---

# rosbash

---

ROS-hoz bash-jellegű parancsok, eredmény stdout-on jelenik meg.

Használatához minden terminálban:

- \$ source /opt/ros/<distro>/setup.bash  
vagy
  - \$ echo "source /opt/ros/<distro>/setup.bash" >> ~/.bashrc
- 
- Tab-kiegészítési lehetőség a ROS parancsokhoz: [Tab Completion](#)
  - További leírás: <https://wiki.ros.org/ROS/Tutorials>

# rosbash – package kezelés

---

- catkin\_create\_pkq: új package létrehozása

- catkin\_make: build package

- rospack: Eszköz a package-ek kezeléséhez. Pl.

rospack find [package\_name]: megkeresi a package\_name nevű csomagot

rospack list: package lista kiírása

rospack depends [package\_name]: listázza a package függőségeit

rospack depends-on [package\_name]: listázza a package-től függő package-eket

- rosstack: stack-hez kapcsolódó információk lekérdezése. Pl:

rosstack list: stack-ek listázása

rosstack contains [package\_name]: megadja, hogy melyik stack-ben található a package

# rosbash – package kezelés

---

- roscd package könyvtár váltás

`roscd [locationname[ /subdir ] ]`: megadott könyvtárra, alkönyvtárra ugrik  
`roscd: $ROS_ROOT`-ban megadott helyre ugrik

- rosls: ROS package fájljainak listázása

`rosls [locationname[ /subdir ] ]`: könyvtár, alkönyvtár tartalmát listázza

- rosdep: ROS függőségek kezelése

`rosdep install [package]`: package függőségek telepítése

# rosbash

---

- **roscp**: fájl másolás egy package-ból
- **rosed**: fájl szerkesztése
- **rosrun**: package futtatható állományának (node) futtatása  
`rosrun [package_name] [node_name]`, pl.  
`rosrun turtlesim turtlesim_node` : turtlesim\_node futtatása  
`rosrun turtlesim turtlesim_node __name:=my_turtle` : turtlesim\_node futtatása my\_turtle néven
- **roscore**: roscore futtatása (ROS rendszermag: master + rosout + parameter server)
- **rosnode**: információ node-okról, pl.  
`rosnode list`: aktív node-ok listázása  
`rosnode info [ros_node]`: információ a node-ról

# rosbash

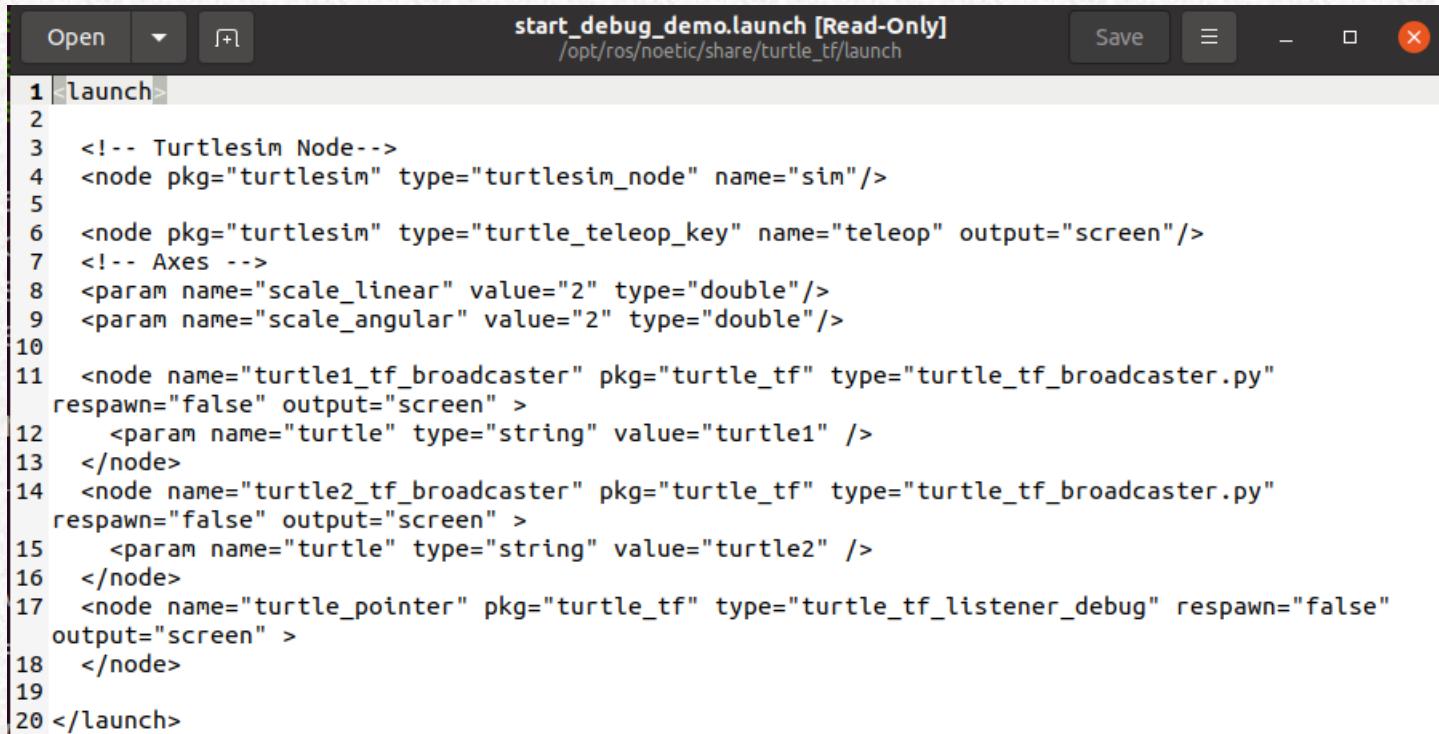
---

- **rostopic**: ROS topic-okról ad információt. pl  
`rostopic list`: topic-ok listázása  
`rostopic echo [topic]`: topic üzeneteit listázza  
`rostopic type [topic]`: megadja a topic típusát  
`rostopic pub [topic] [msg_type] [args]`: publish  
`rostopic hz [topic]`: pubilkálás, kiolvasás gyakoriságáról ad infót
- **rosmsg**: információ az üzenetekről  
`rosmsg show [message]`: üzenet típusát adja meg
- **rosservice**: ROS service listázás, meghívás  
`rosservice list`: aktív service-ek kilistázása  
`rosservice call`: service meghívása
- **rosparam**: ROS paraméterek kezelése

# rosbash

---

- `roslaunch`: ROS node-ok indítása a launch fájl alapján  
`roslaunch [package] [filename.launch]`



The screenshot shows a code editor window with the following details:

- Title Bar:** `start_debug_demo.launch [Read-Only]`  
/opt/ros/noetic/share/turtle\_tf/launch
- Buttons:** Open, Save, Minimize, Maximize, Close
- Code Content:** A ROS launch file with line numbers 1 through 20. The code defines several nodes and parameters for a turtle simulation.

```
1 <launch>
2
3   <!-- Turtlesim Node-->
4   <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
5
6   <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>
7   <!-- Axes -->
8   <param name="scale_linear" value="2" type="double"/>
9   <param name="scale_angular" value="2" type="double"/>
10
11  <node name="turtle1_tf_broadcaster" pkg="turtle_tf" type="turtle_tf_broadcaster.py"
12    respawn="false" output="screen" >
13    <param name="turtle" type="string" value="turtle1" />
14  </node>
15  <node name="turtle2_tf_broadcaster" pkg="turtle_tf" type="turtle_tf_broadcaster.py"
16    respawn="false" output="screen" >
17    <param name="turtle" type="string" value="turtle2" />
18  </node>
19  <node name="turtle_pointer" pkg="turtle_tf" type="turtle_tf_listener_debug" respawn="false"
20    output="screen" >
</node>
</launch>
```