Laurent LEQUIEVRE – Ingénieur CNRS
Institut Pascal UMR6602 http://ip.univ-bpclermont.fr/index.php/fr/
Equipe Maccs http://ip.univ-bpclermont.fr/index.php/fr/maccs
Juan Antonio CORRALES RAMON
Youcef MEZOUAR

Introduction à ROS
Master recherche robotique

Laurent LEQUIEVRE – Ingénieur CNRS
Institut Pascal UMR6602 http://ip.univ-bpclermont.fr/index.php/fr/
Equipe Maccs http://ip.univ-bpclermont.fr/index.php/fr/maccs
Juan Antonio CORRALES RAMON
Youcef MEZOUAR

cd ~/Bureau
git clone https://github.com/lequievre/tp_ros_master_robotique.git

Index

- C'est quoi ROS ?
- Pourquoi ROS ?
- Histoire de ROS
- Quelques avantages de ROS
- Quelques applications sous ROS
- Quelques éléments de base
- Jouons un peu avec ROS - turtlesim
- La communauté ROS

C'est quoi ROS ?

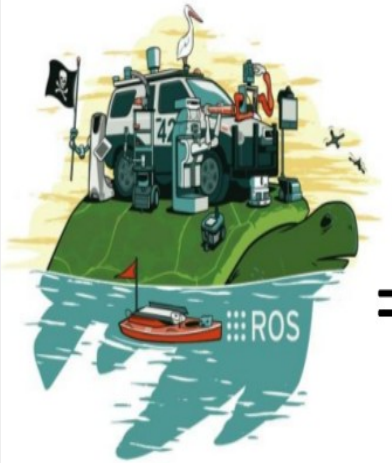
http://www.ros.org/

- ROS est l'acronyme de **R**obot **O**perating **S**ystem
- Un système d'exploitation pour les robots
- Se situe entre le système d'exploitation et le middleware

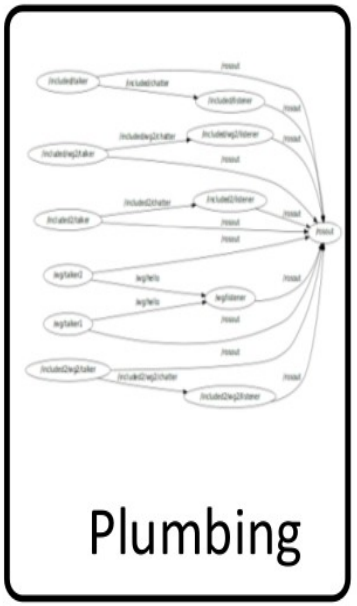
ROS

C'est quoi ROS ?

ROS is...

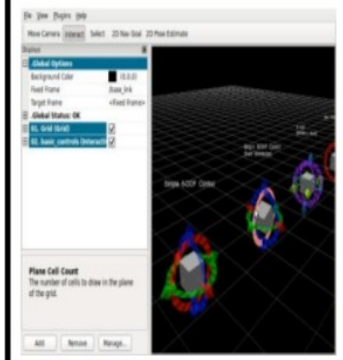


=



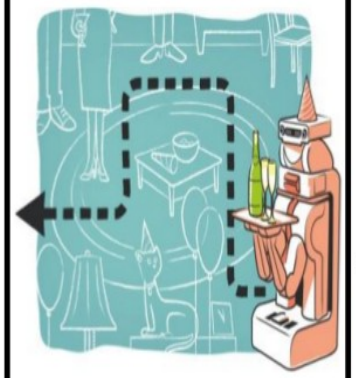
Plumbing

+



Tools

+



Capabilities

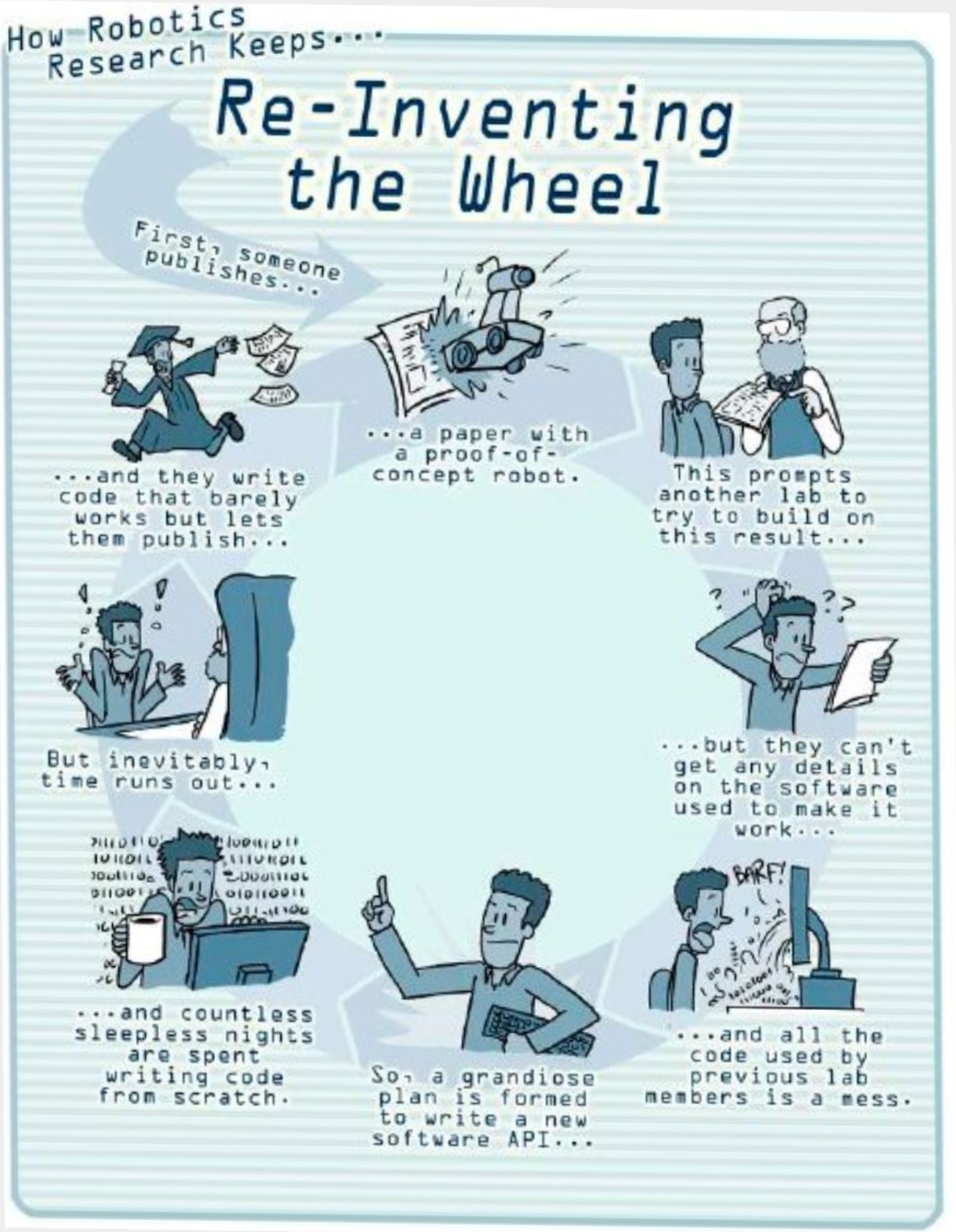
+

Ecosystem

C'est quoi ROS ?

- *Abstraction matérielle* (***ros control***)
- *Gestion de la concurrence* (***ros spin***)
- Communication par messages asynchrones/synchrones (***ros topics/services***)
- Gestion de packages (***catkin***), serveur de paramètres (***ros param***)
- Transformations géométriques (***tf***), langage de description d'un robot (***URDF***)
- Simulation (***Gazebo***), planification de mouvement (***MoveIt!***), visualisation (***Rviz***)
- Framework pour interfaces graphiques (***Rqt***)
- Utilisable en mode commande (***rosrun, roslaunch, roscd, rosfind ...***)

Pourquoi ROS ?



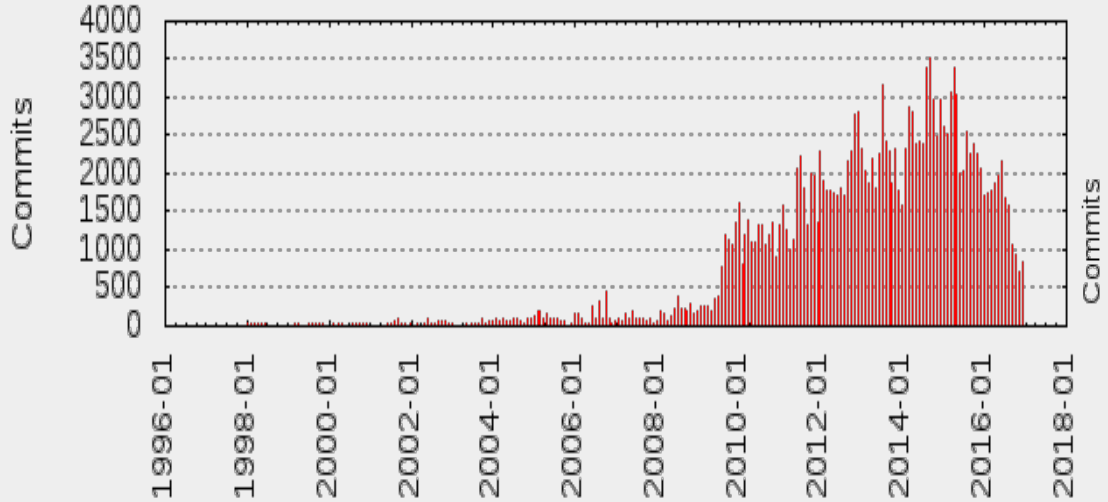
Histoire de ROS

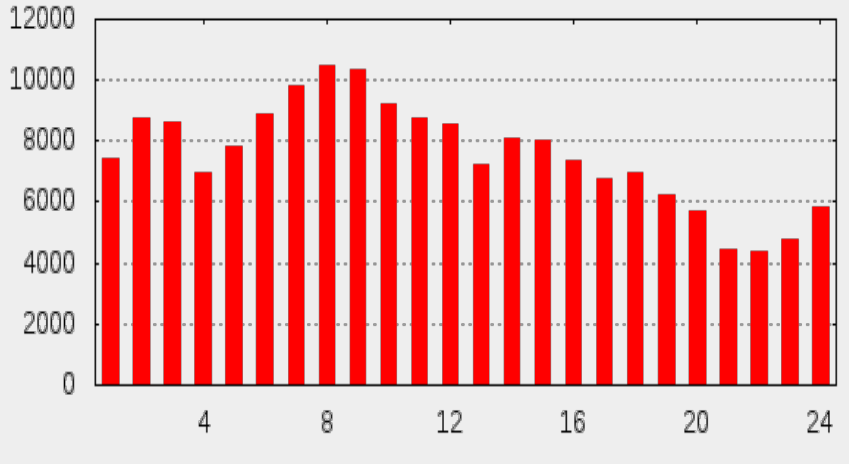
- Initialement développé en 2007 par le **Stanford Artificial Intelligence Laboratory** dans le projet STAIR (**ST**anford **A**rtificial **I**ntelligence **R**obot) (sous le nom *switchyard*).http://stair.stanford.edu/index.php
- Entre 2008 et 2013, le développement a continué par les fortes contributions de **Willow Garage** pour son robot humanoïde PR2.
- Sortie de la version ROS 1.0 en Janvier 2010.
- Depuis 2013, ROS dépend de l'**O**pen **S**ource **R**obotics **F**oundation. Organisation indépendante à but non lucratif, dont sa mission est de supporter le développement, la distribution de logiciels libres pour la robotique. http://www.osrfoundation.org/
- La communauté scientifique s'est aussi emparée de ROS et contribue à le développer. On observe une augmentation importante de dépôts contenant des paquets ROS dans le monde.

Histoire de ROS

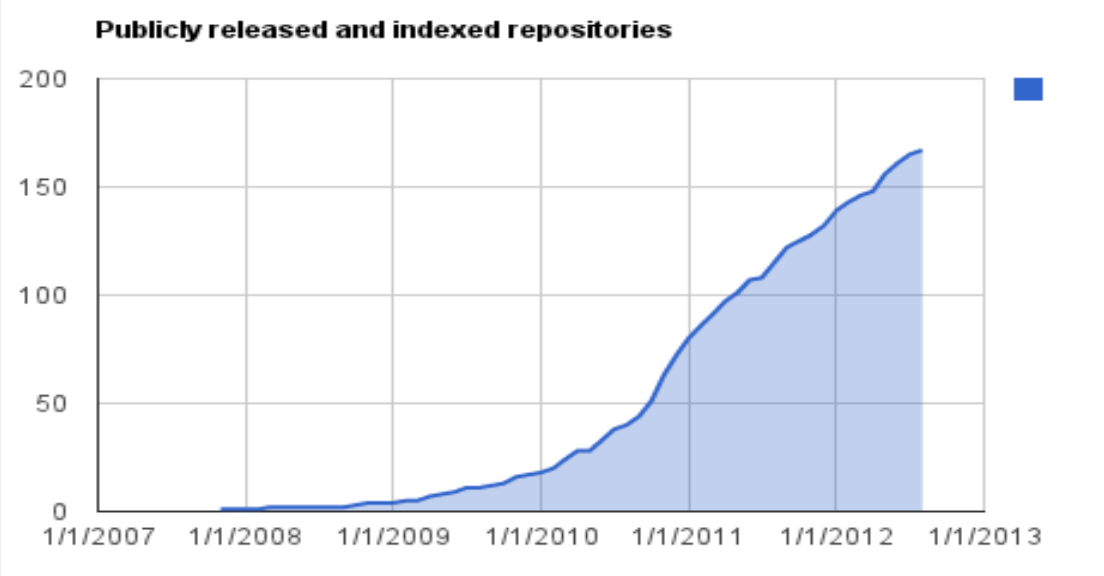
Quelques statistiques sur la version Indigo Igloo :

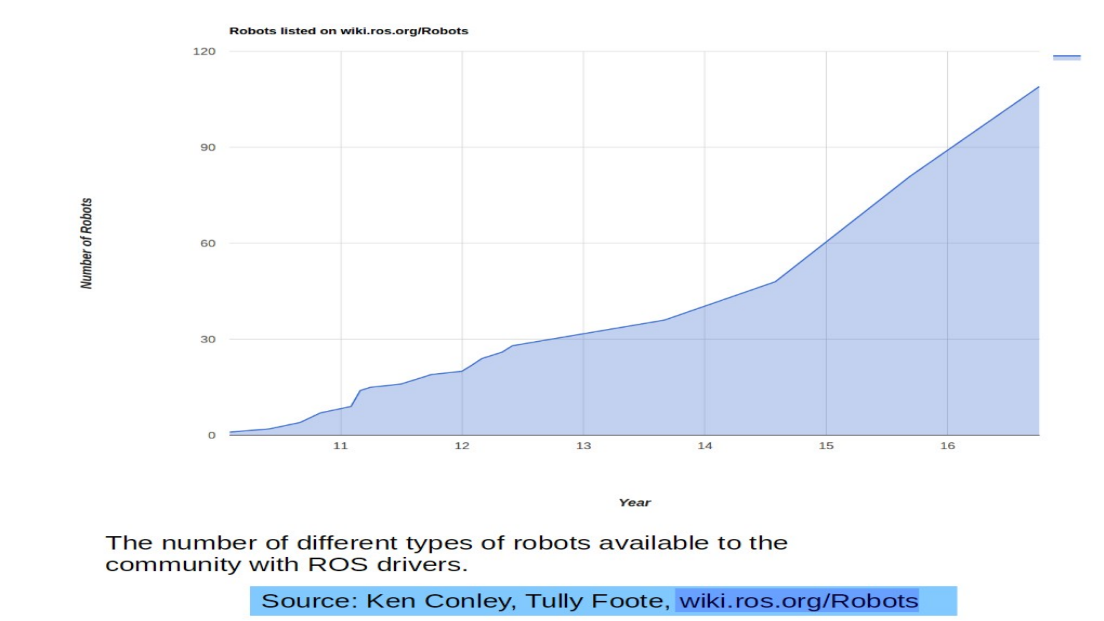
- 14 millions de lignes de code.
- 2 477 auteurs.
- 181 509 'commits'.
- Une moyenne de 73,3 'commits' par auteur.
- Les 'commits' se font sur 24 fuseaux horaires.





Histoire de ROS





The number of different types of robots available to the community with ROS drivers.

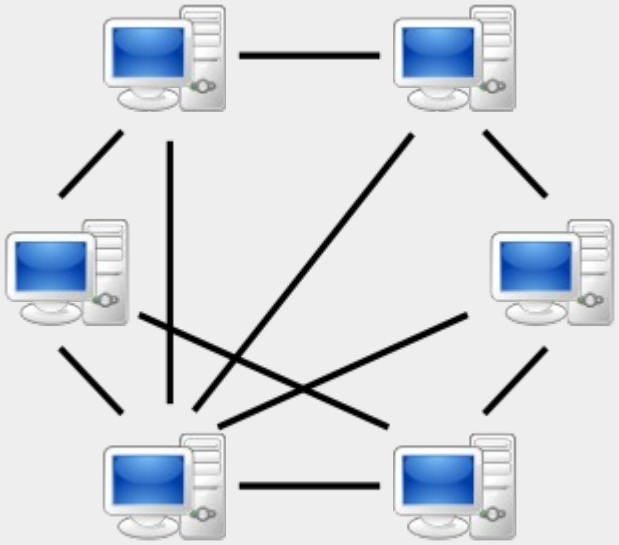
Source: Ken Conley, Tully Foote, wiki.ros.org/Robots

Histoire de ROS

- ROS est sous licence BSD (**B**erkeley **S**oftware **D**istribution) qui est une licence libre utilisée pour la distribution de logiciels.
- C'est une des licences les moins restrictives.
- Elle permet de réutiliser tout ou partie du logiciel sans restriction, qu'il soit intégré dans un logiciel libre ou propriétaire.
- ROS évolue régulièrement, voici liste des versions : http://wiki.ros.org/Distributions
- Liste des robots compatibles avec ROS : **http://robots.ros.org/**
- Nous allons utilisé la version ROS Indigo Igloo qui fonctionne sous Ubuntu 14.04 LTS (Trusty).

Quelques avantages de ROS

- **Peer-to-peer** : cela permet à plusieurs ordinateurs sous différents OS de communiquer via un réseau en échangeant des données, sans transiter par un serveur central. Dans le cas de ROS on parle de 'nœud'.



Quelques avantages de ROS

- **Multi-language** : Les paquets ROS peuvent être développés en différents langages de programmation (C++, Python, Lisp). La structure d'un 'message ROS' est définit dans un langage spécifique IDL (**I**nterface **D**escription **L**anguage) pour être ensuite traduit automatiquement dans un langage de programmation cible.
- **Basé sur des outils :** Ros n'est pas monolithique, mais plutôt basé sur un design 'microkernel'. Il est composé de plein de petits outils pour développer, compiler et exécuter les différents composants de ROS. Ce qui lui confère plus de robustesse et d'évolution qu'un système centralisé.

Quelques avantages de ROS

- **Gratuit et Open-source** : Tous les sources sont disponibles, la plupart des paquets ROS ont la licence BSD.

- **Léger** : Le système ROS a été conçu et développé de manière modulaire, toute sa complexité est incluse dans des librairies externes, ce qui le rend léger.

- ROS **utilise aussi du code issus d'autres librairies 'open source'** comme par exemple : OpenRave, OpenCV …

Quelques applications sous ROS

- Enregistrement et visualisation des données : **rosbag**

http://wiki.ros.org/rosbag

- Visualisation et simulation : **rviz** (3D), **gazebo** (3D)

http://wiki.ros.org/rviz

http://wiki.ros.org/gazebo_ros_pkgs

- Drivers : **camera_drivers, laser_drivers , imu_drivers (*Inertial Measurement Unit)***

http://wiki.ros.org/camera_drivers

http://wiki.ros.org/laser_drivers

http://wiki.ros.org/imu_drivers

Quelques applications sous ROS

- 3D processing : **perception_pcl** (Point Cloud Library), **laser_pipeline**

http://wiki.ros.org/perception_pcl

http://wiki.ros.org/laser_pipeline

- Image processing (2D) : **vision_opencv** (OpenCV), **visp**

http://wiki.ros.org/vision_opencv

http://wiki.ros.org/visp

- Transformations : **tf**, **tf_conversions**

http://wiki.ros.org/tf

http://wiki.ros.org/tf_conversions

Quelques applications sous ROS

- Navigation (odometry, ego-motion, SLAM) : **navigation**

http://wiki.ros.org/navigation

- Controllers (position, force, speed, transmissions) : **ros_control**

http://wiki.ros.org/ros_control

- Robot modelling : **urdf** (XML description of robots).

http://wiki.ros.org/urdf

Quelques applications sous ROS

- Motion planning : **MoveIt!** (library **OMPL**), **OpenRAVE**.

http://moveit.ros.org/

http://wiki.ros.org/openrave

- Grasping and manipulation : **GraspIt!, OpenRAVE (grasping Module)**.

http://wiki.ros.org/graspit

http://openrave.org/docs/0.6.6/openravepy/databases.grasping/

Quelques éléments de base

- Le **package** : L'unité principale d'organisation logicielle de ROS. C'est un répertoire contenant des **nœuds**, des fichiers de configurations, des librairies externes.

```
.
..
CMakeLists.txt
include
kuka_lwr_controllers_plugins.xml
msg
package.xml
src
```

*Le fichier **package.xml** fournit les méta-informations sur le package.*
*Le fichier **CMakeLists.txt** contient les ordres de compilation.*
*Le dossier **src** contient les sources du package.*
*Le dossier **include** contient les entêtes du package.*
*Le dossier package doit être dans le **ROS_PACKAGE_PATH***

Quelques éléments de base

- Le **package** (suite) :

La commande ***rospack*** : Donne des infos sur les paquets.

```
rospack find my_super_package

rospack depends1 my_super_package

rospack depends my_super_package
```

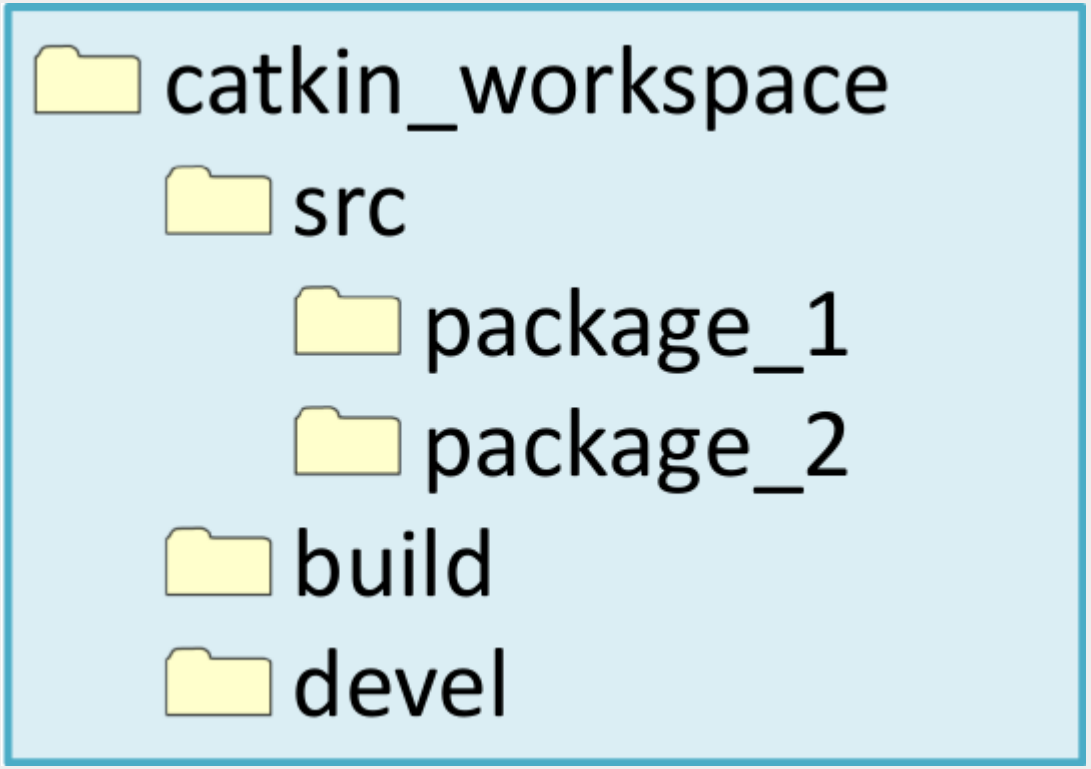
La commande ***roscd*** : Bascule le répertoire directement sur le paquet.

```
roscd my_super_package
```

La commande **catkin_create_pkg** permet de créer un nouveau package.

Quelques éléments de base

- Le **workspace** : C'est un répertoire contenant plusieurs packages ROS.



Le dossier ***build*** contient les détails de la compilation des packages.
Le dossier ***devel*** contient les résultats compilés (les cibles du cmake).

Quelques éléments de base

Etapes de création d'un workspace catkin

1. mkdir -p ~/catkin_ws/src
2. cd ~/catkin_ws/src
3. catkin_init_workspace
4. cd ~/catkin_ws/
5. catkin_make
6. source devel/setup.bash

- La commande **catkin_init_workspace** permet d'initialiser le **workspace** en créant un fichier **CMakeLists.txt** dans le dossier **src**.
- La commande **catkin_make** crée les dossiers **build** et **devel**.
- La commande **source devel/setup.bash** ajoute le chemin **~/catkin_ws/src** dans la variable **ROS_PACKAGE_PATH**.

Quelques éléments de base

- Le **stack of packages** : Une collection de packages, c'est un répertoire qui contient des répertoires de packages.

  Permet de rassembler des packages interdépendants autour d'une même thématique.

  ROS Navigation Stack
  https://github.com/ros-planning/navigation/tree/indigo-devel

Quelques éléments de base

- Le **message** : Types de données ROS, échangés via un ***topic*** entre des nœuds.
- Il est définit dans un langage IDL (**I**nterface **D**escription **L**anguage).
- Sauvegardé dans un fichier ***.msg*** dans le sous dossier ***msg*** du package.
- Génération automatique de code via des instructions dans le CMakeLists.txt et package.xml.

Quelques éléments de base

- Types disponibles dans les messages : http://wiki.ros.org/msg#Fields
- On peut aussi ajouter des constantes. Ex : ***int32 X=12***
- La commande ***rosmsg*** permet d'afficher les infos d'un message.

```
rosmsg show RPY
```

RPY.msg
float64 roll
float64 pitch
float64 yaw

PoseRPY.msg
#include <my_package/RPY.h>
geometry_msgs/Vector3 position
RPY orientation

Quelques éléments de base

- Le **node (noeud)** : Un ***node*** est un exécutable qui utilise ROS pour communiquer via des ***topics*** ou des ***services*** avec d'autres nœuds.
- Les ***nodes*** peuvent être distribués sur plusieurs machines séparées.
- Les ***nodes*** sont programmés via une librairie cliente (roscpp, rospy, roslisp) en fonction du langage de programmation utilisé.

```
rosnode info my_node

rosrun my_package my_node
```

rosnode permet d'obtenir des infos sur un node.
rosrun permet de lancer un node qui se trouve dans un package.

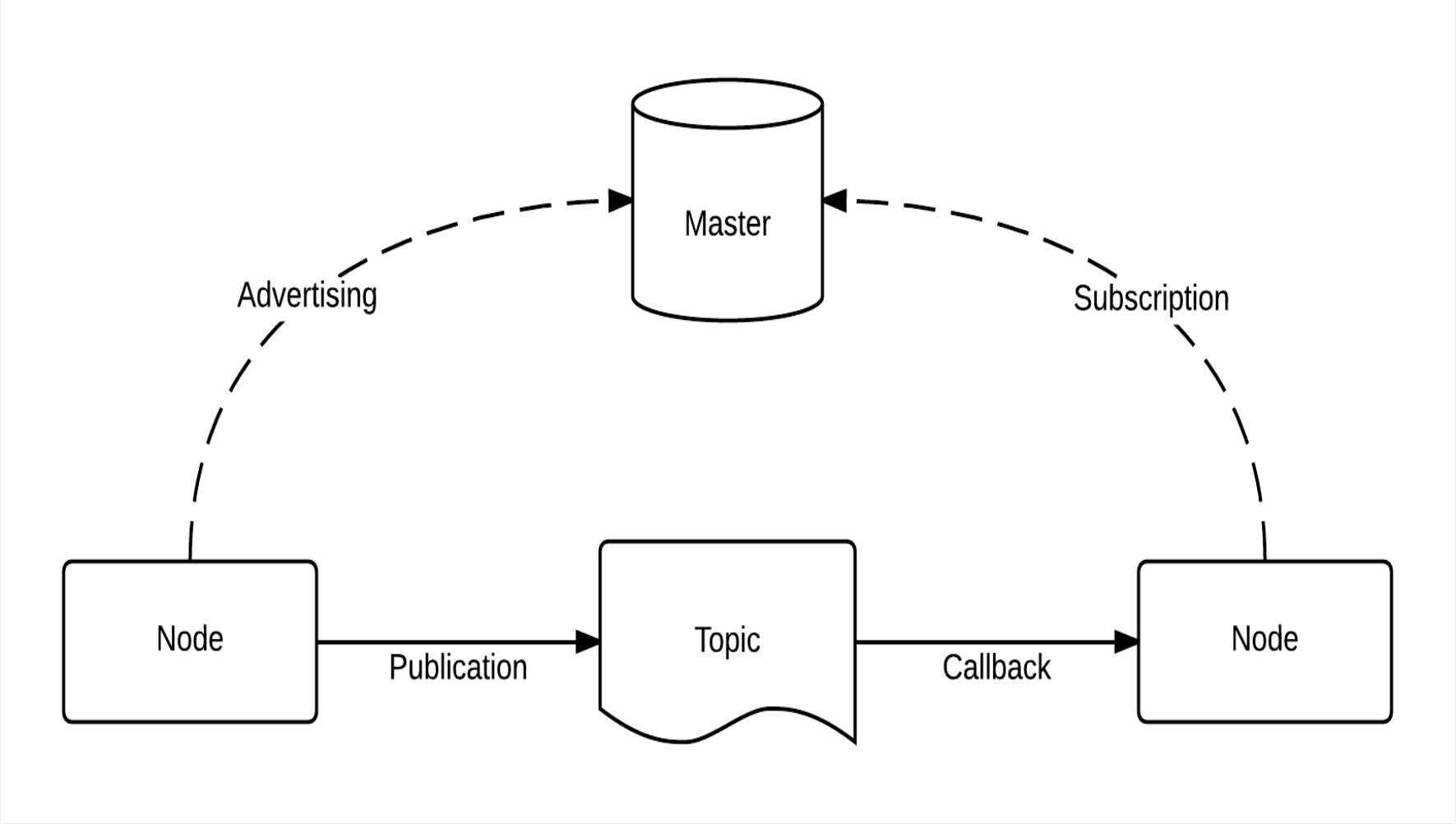
Quelques éléments de base

- Le **Master** : Aide les nœuds à se trouver mutuellement (similaire à un DNS).
- Permet la communication ***peer-to-peer*** entre les nœuds. Il mémorise les ***topics*** et ***services*** de chaque nœuds.
- On le lance via la commande ***roscore***.

Quelques éléments de base

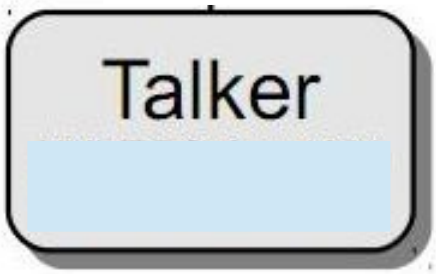
- Le **Topic** : Une sorte de bus qui possède un nom et permet aux nœuds de publier des messages sur un ***topic*** aussi bien que souscrire à un ***topic*** pour recevoir des messages.
- Il s'agit d'une communication asynchrone.
- Les ***Publishers*** sont des nœuds qui publient des messages dans un ***topic***.
- Les ***Subscribers*** sont des nœuds qui reçoivent des messages via un ***topic***.

Quelques éléments de base

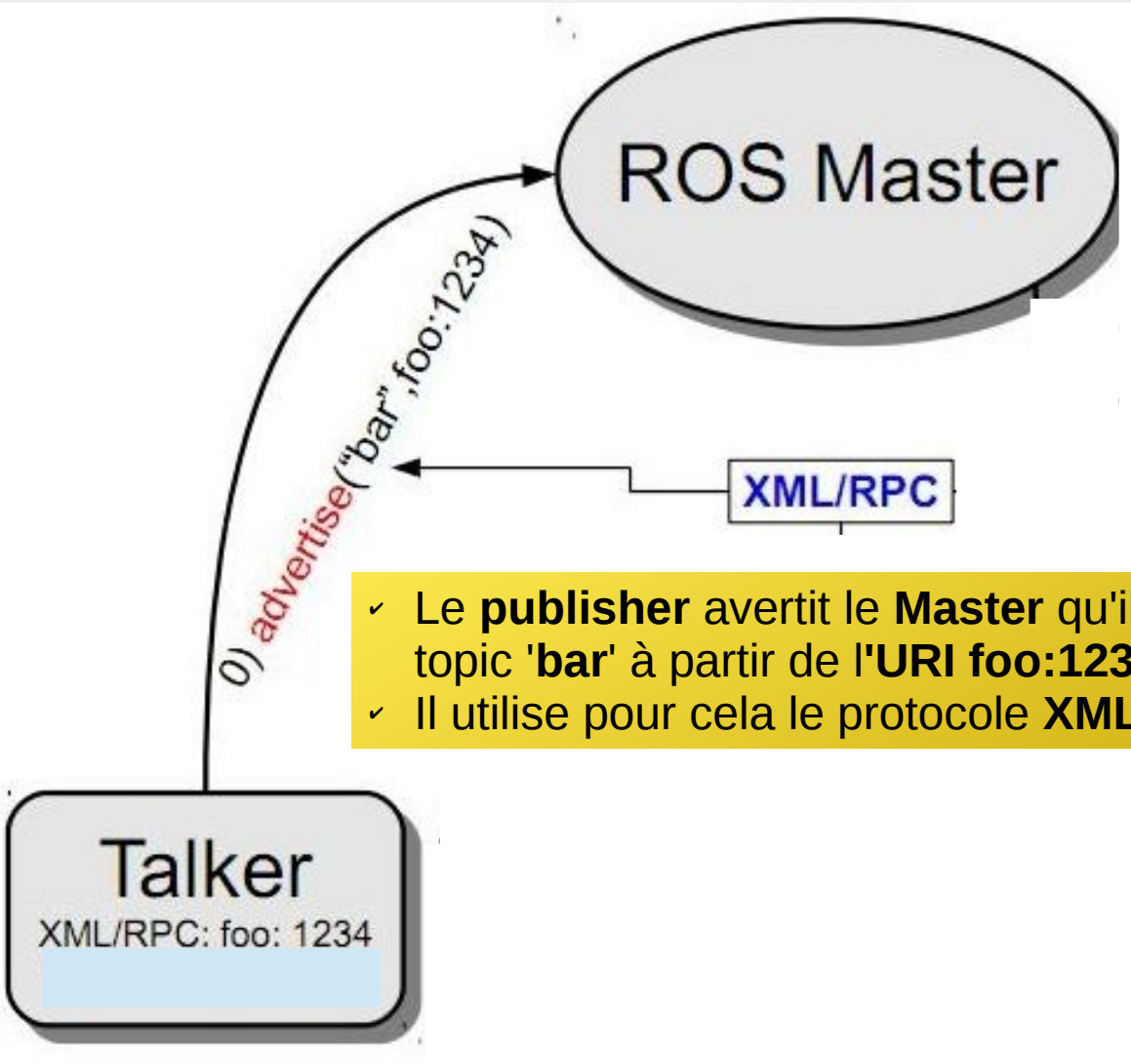


Quelques éléments de base

Un noeud de type **Master**
Un nœud qui publie **Talker : (*Publisher*)**
Un nœud qui écoute **Listener : (*Subscriber*)**

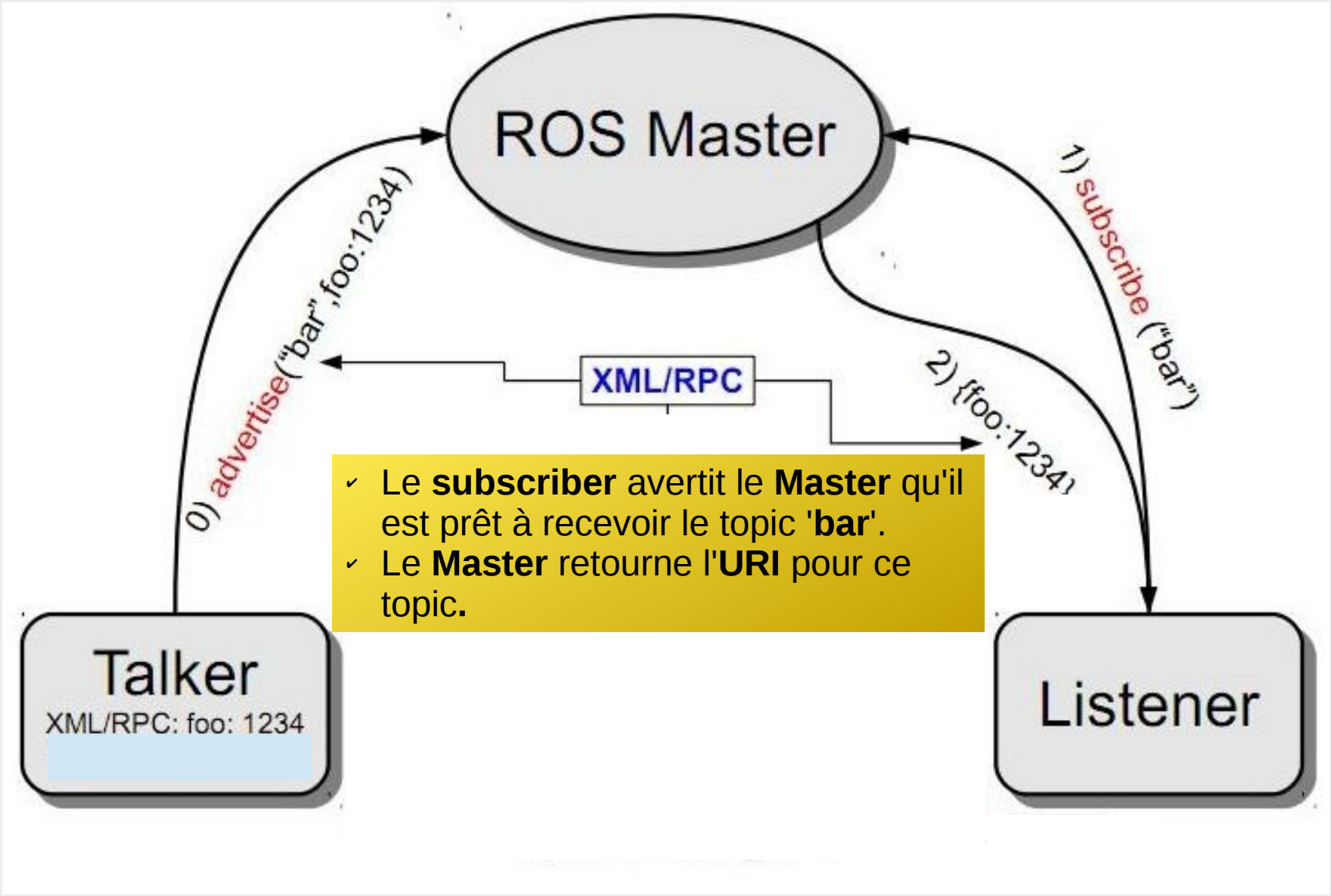


Quelques éléments de base



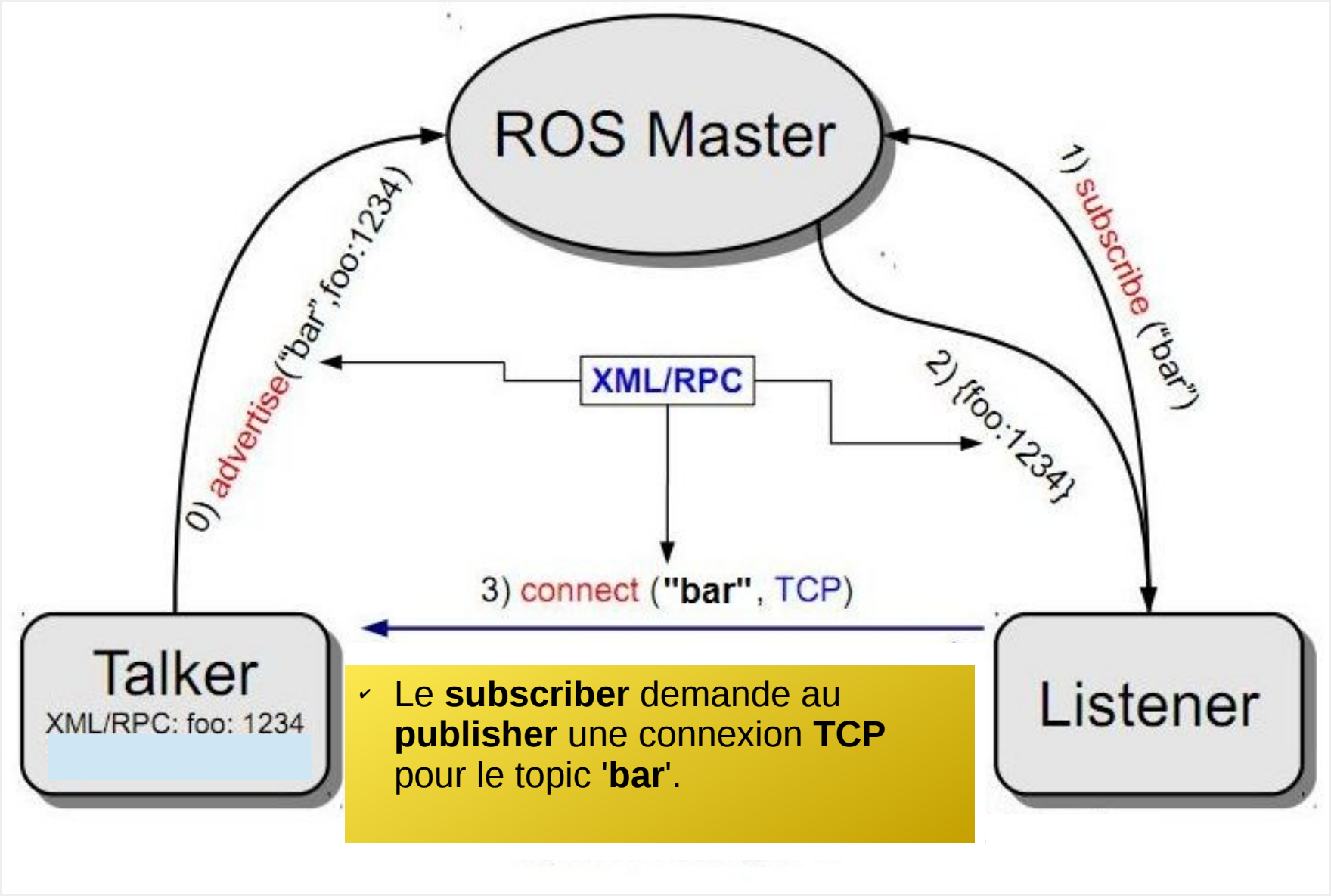
- Le **publisher** avertit le **Master** qu'il est prêt à publier sur le topic '**bar**' à partir de l'**URI foo:1234.**
- Il utilise pour cela le protocole **XML/RPC.**

Quelques éléments de base



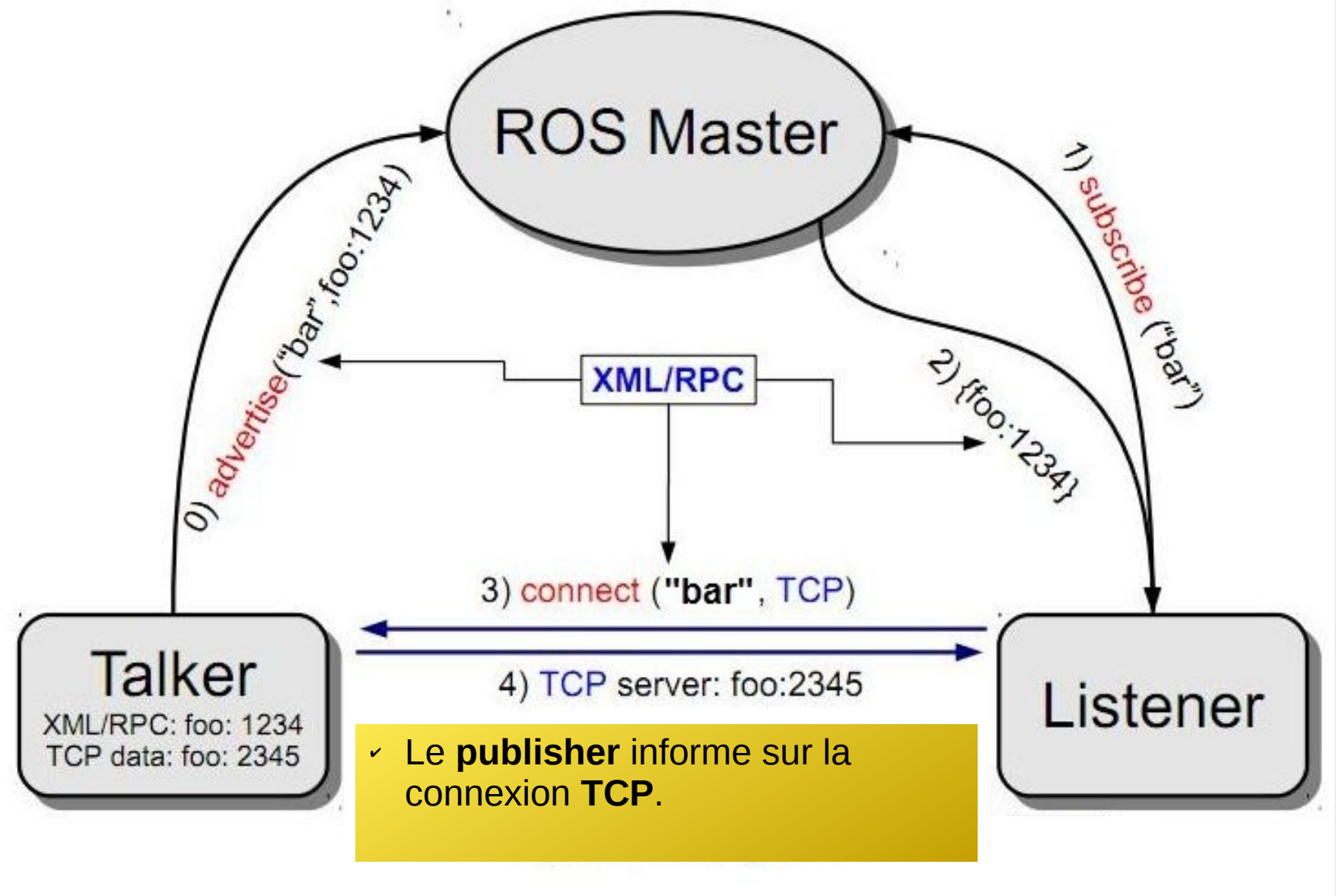
- Le **subscriber** avertit le **Master** qu'il est prêt à recevoir le topic '**bar**'.
- Le **Master** retourne l'**URI** pour ce topic**.**

Quelques éléments de base



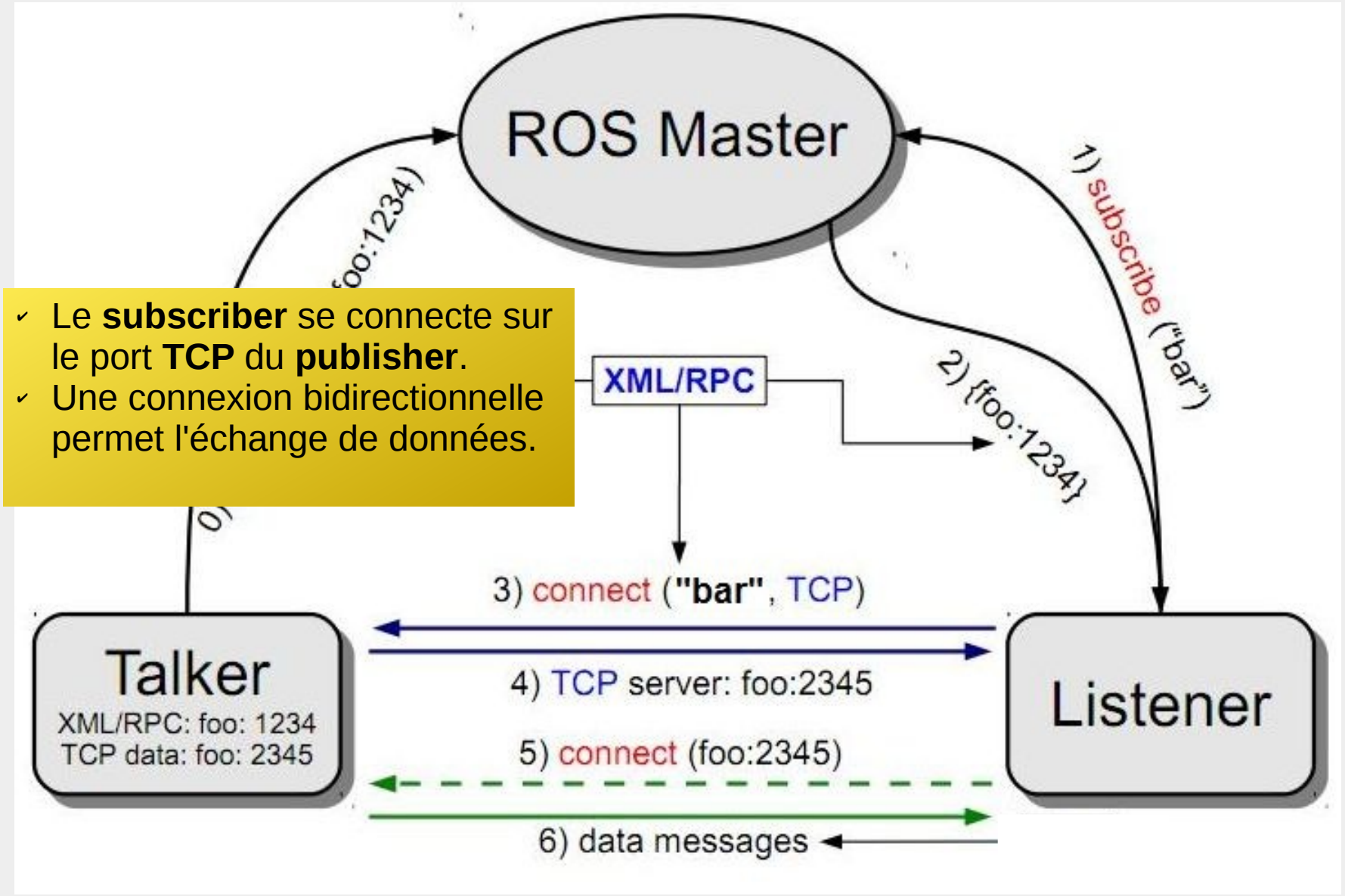
- Le **subscriber** demande au **publisher** une connexion **TCP** pour le topic '**bar**'.

Quelques éléments de base



- Le **publisher** informe sur la connexion **TCP**.

Quelques éléments de base

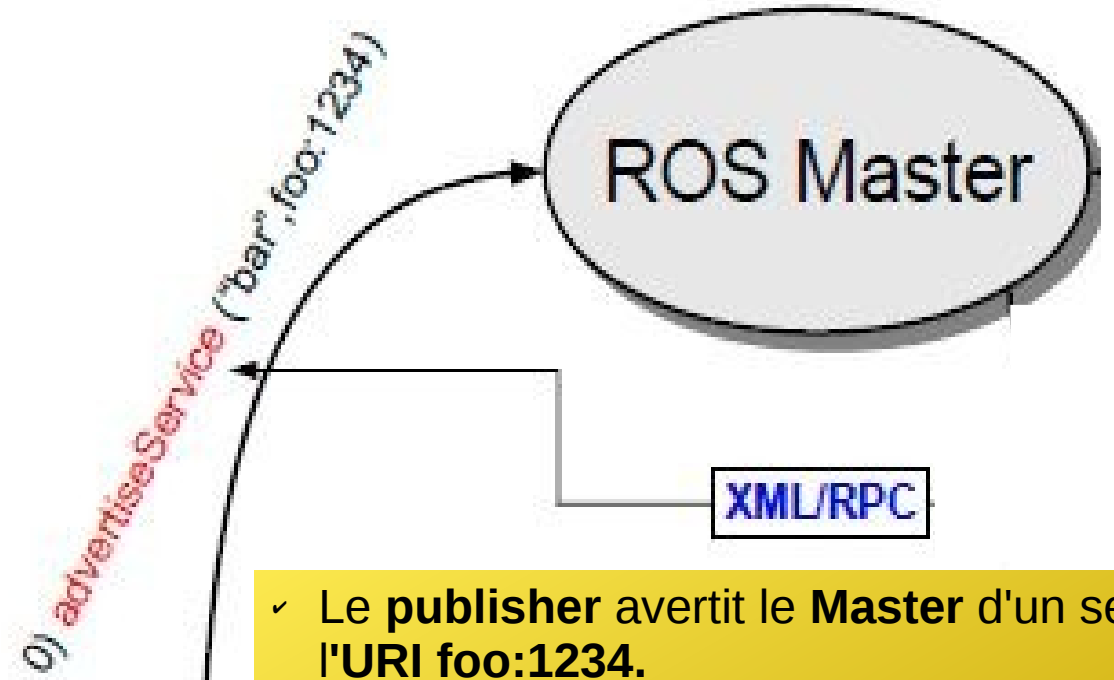


- Le **subscriber** se connecte sur le port **TCP** du **publisher**.
- Une connexion bidirectionnelle permet l'échange de données.

Quelques éléments de base

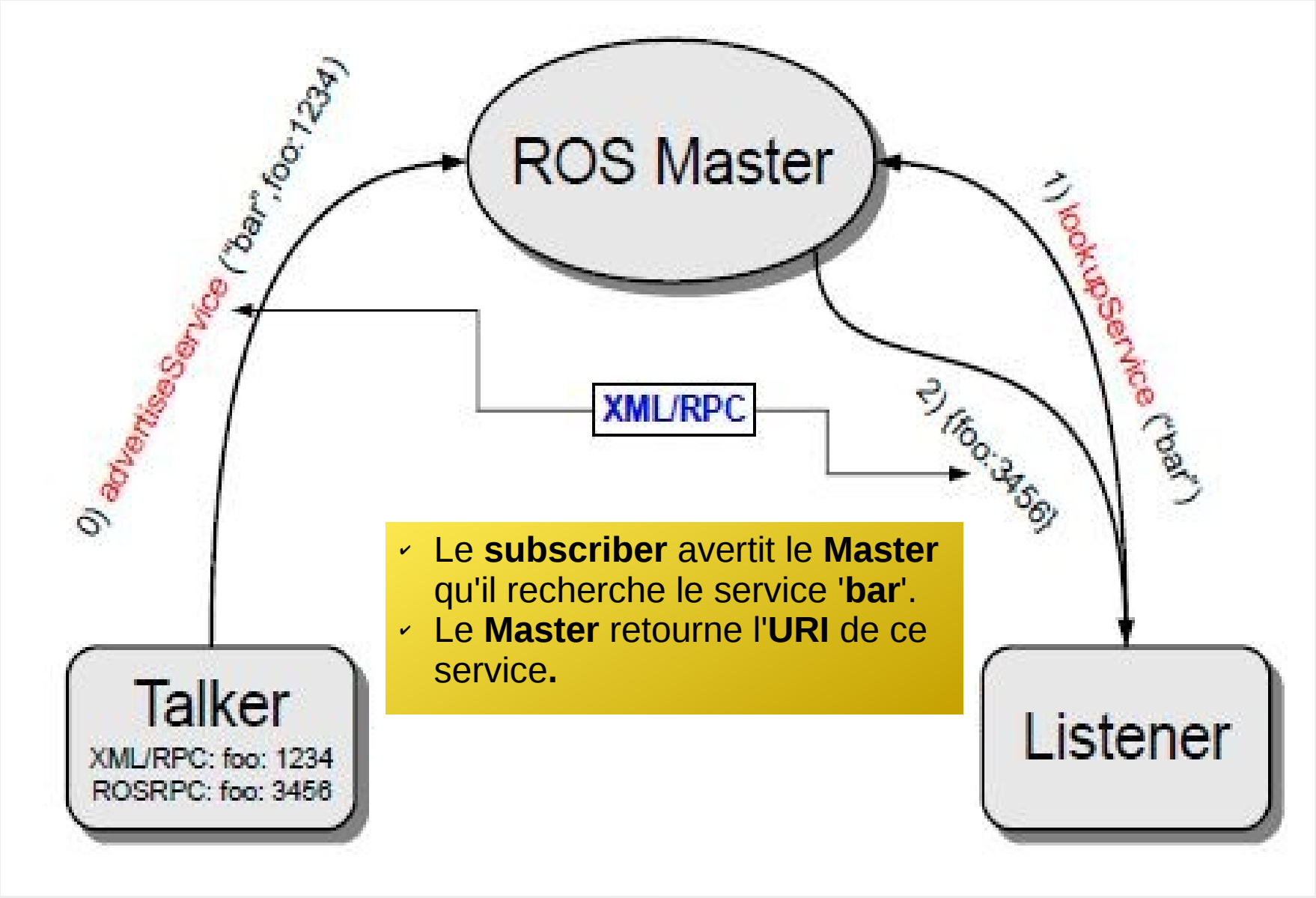
- Le **Service** : C'est une méthode de communication synchrone entre 2 nœuds (RPC = Remote Procedure Calls).
- Ce coup-ci un 'node 1' peut envoyer une demande ('request') à un autre 'node 2', le 'node 1' attendra une réponse ('response') du 'node 2'.
- C'est en quelque sorte un appel 'remote' de fonctionnalités d'un 'node' à partir d'un autre 'node'.
- Les services sont décrits dans le langage IDL dans un fichier à l'extension ***.srv*** dans le sous dossier ***srv*** du package.
- Génération automatique de code via des instructions dans le CMakeLists.txt et package.xml.

Quelques éléments de base



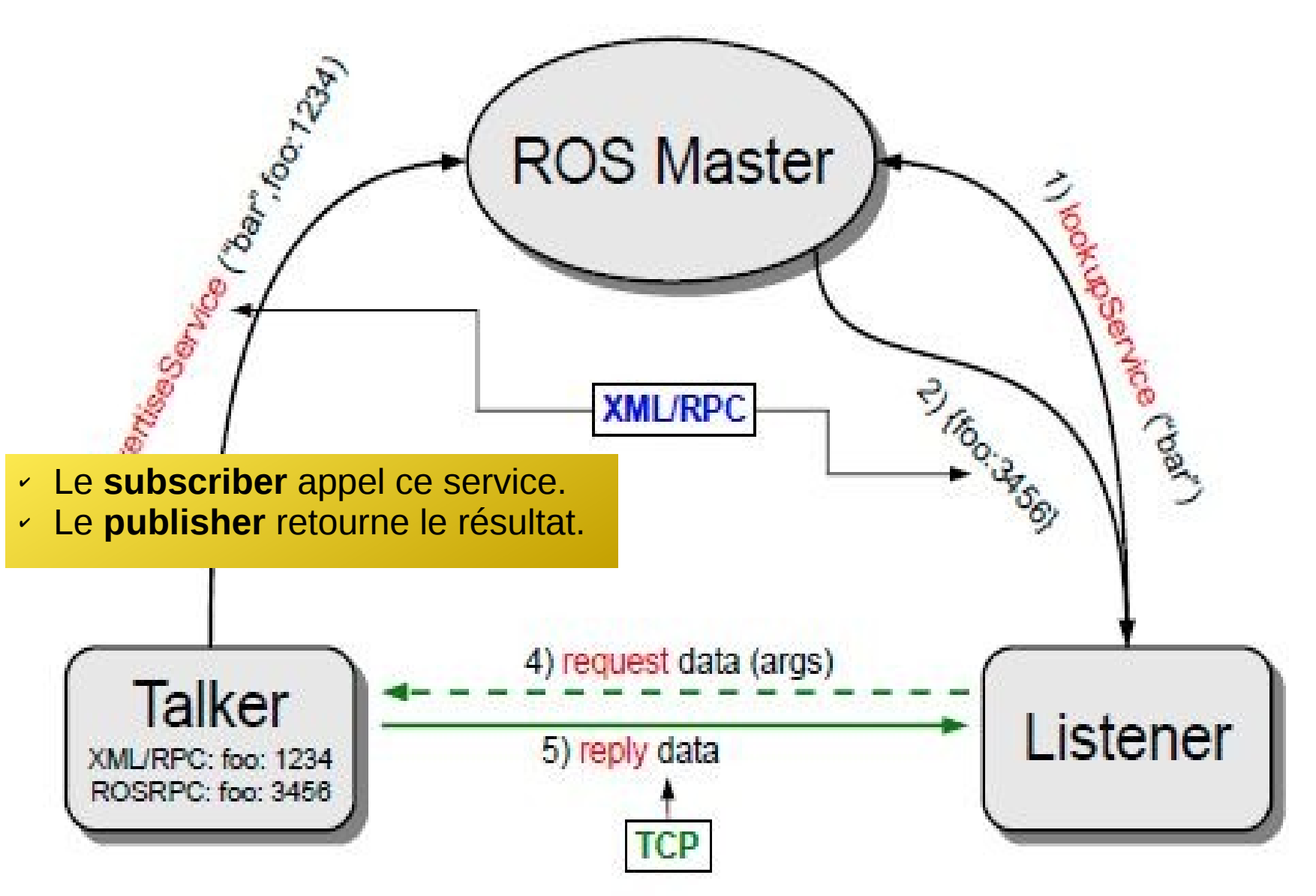
- Le **publisher** avertit le **Master** d'un service '**bar**' à partir de l'**URI foo:1234.**
- Il utilise pour le cela le protocole **XML/RPC.**

Quelques éléments de base



- Le **subscriber** avertit le **Master** qu'il recherche le service '**bar**'.
- Le **Master** retourne l'**URI** de ce service**.**

Quelques éléments de base

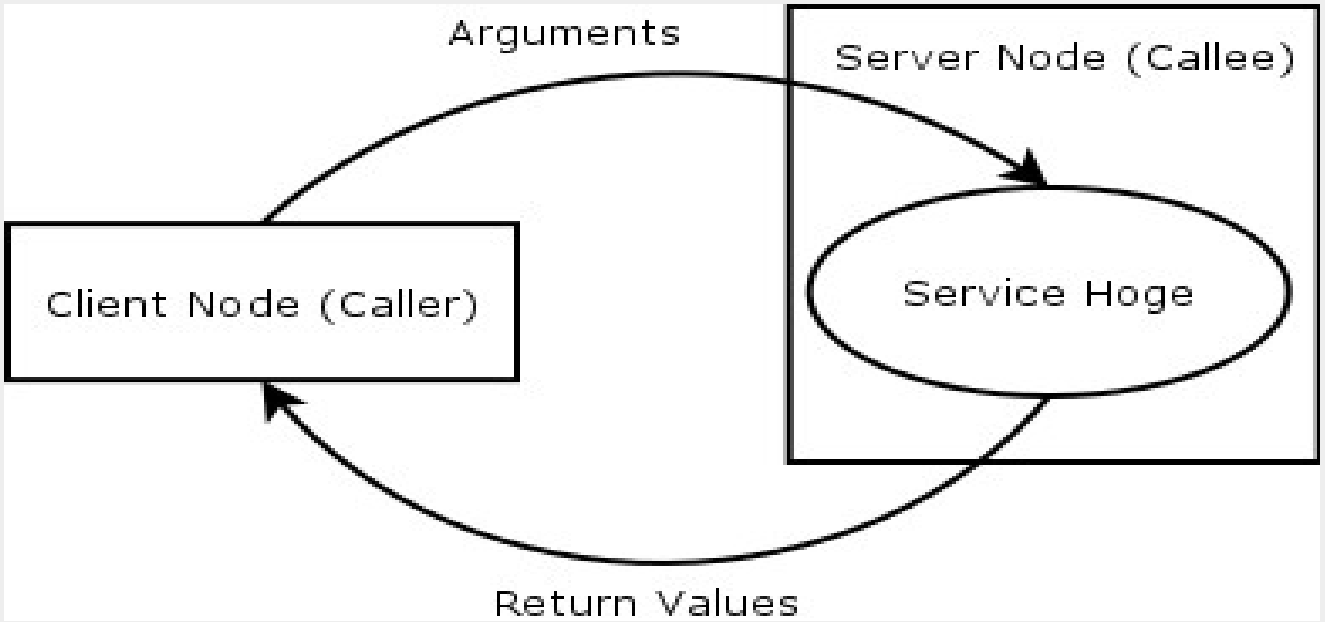


- Le **subscriber** appel ce service.
- Le **publisher** retourne le résultat.

Quelques éléments de base

Request → a,b Response → sum (séparés par '–-') :

add_two_ints.srv
int64 a
int64 b
---
int64 sum



Quelques éléments de base

- La commande ***rosservice*** permet de lister, trouver, lancer un service.

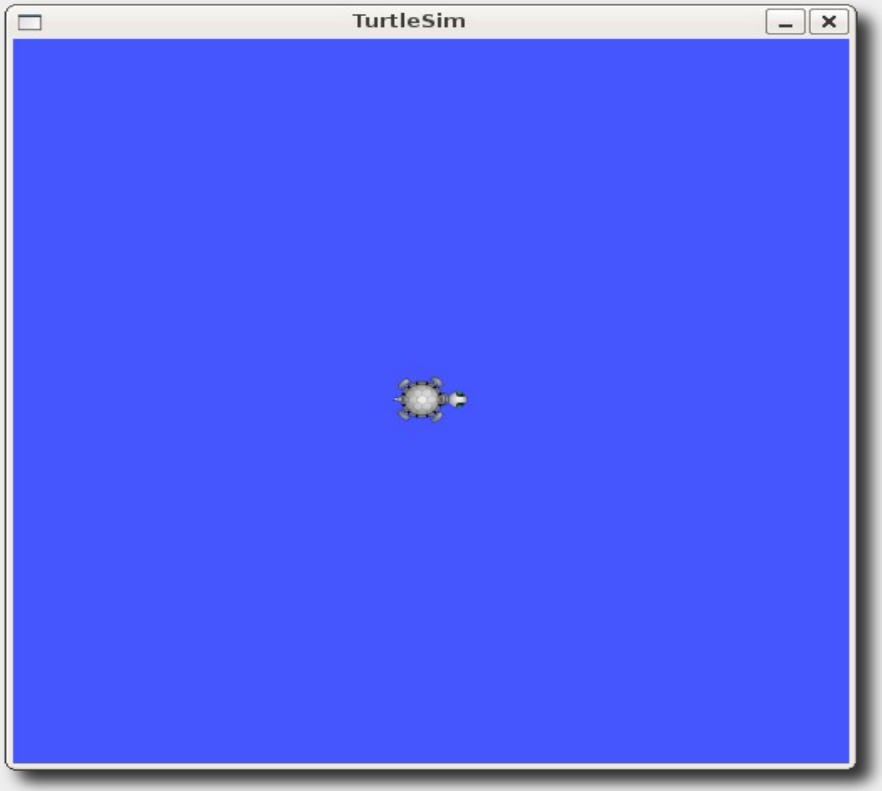
- La commande ***rossrv*** permet d'afficher les infos d'un service.

```
rosservice call /add_two_ints 1 2

rossrv show /add_two_ints
```

Jouons un peu avec ROS - turtlesim

http://wiki.ros.org/turtlesim



Lancer le **Master**

```
roscore
```

Lancer le **publisher**

```
rosrun turtlesim turtle_teleop_key
```

Lancer le **subscriber**

```
rosrun turtlesim turtlesim_node
```

Lister les **Noeuds** lancés, obtenir des infos

```
rosnode list
rosnode info /teleop_turtle
rosnode ping /teleop_turtle
```

Jouons un peu avec ROS - turtlesim

http://wiki.ros.org/turtlesim

Lancer un noeud **sous un autre nom**

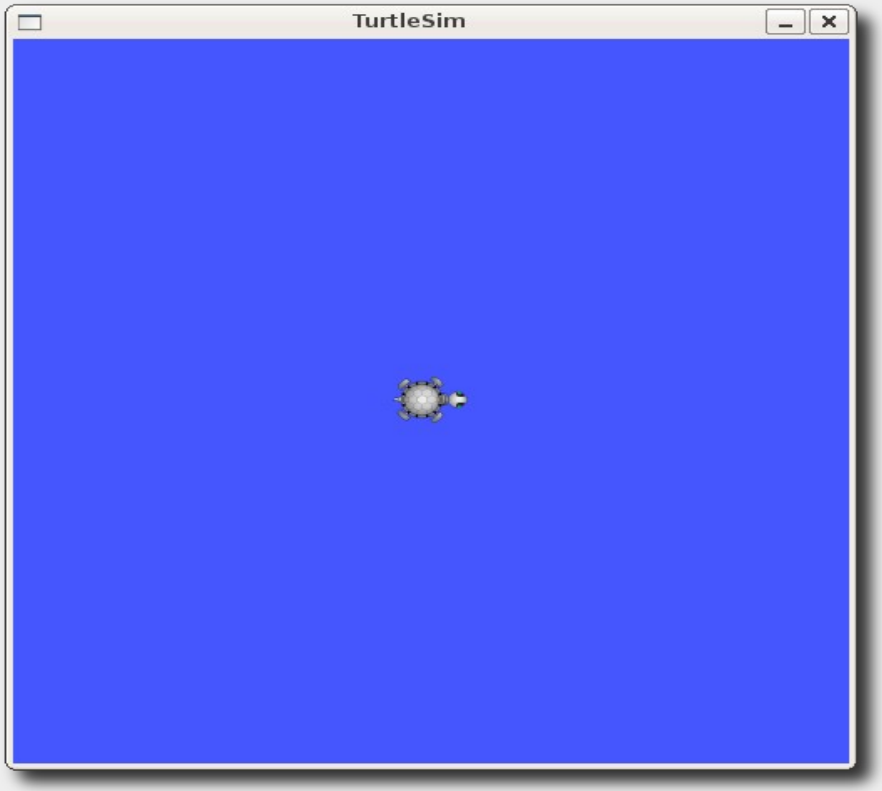
```
rosrun turtlesim turtle_teleop_key __name :=Toto
```

Lister les **Noeuds** lancés, obtenir des infos

```
rosnode list
rosnode info /Toto
rosnode ping /Toto
```

Jouons un peu avec ROS - turtlesim

http://wiki.ros.org/turtlesim



Voir le graphe ROS :

rqt_gaph

rosrun rqt_graph rqt_graph

Jouons un peu avec ROS - turtlesim

Liste des topics :

```
rostopic list
```

Infos sur le topic de commande :

```
rostopic info /turtle1/cmd_vel
```

Infos sur le message du topic de commande :

```
rosmsg show geometry_msgs/Twist
```

Appuyer sur les flèches de votre clavier et afficher les données du topic commande :

```
rostopic echo /turtle1/cmd_vel
```

Jouons un peu avec ROS - turtlesim

Publier des données sur le topic commande :

Publier une seule fois (option -1) :

rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'

Ou encore comme ceci :

rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist '{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}'

Publier à une fréquence (rate) de 1hz (option -r 1) :

rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'

Jouons un peu avec ROS - turtlesim

Infos sur le topic pose :

```
rostopic info /turtle1/pose
```

Infos sur le message du topic pose :

```
rosmsg show turtlesim/Pose
```

Appuyer sur les flèches de votre clavier et afficher les données du topic pose :

```
rostopic echo /turtle1/pose
```

Jouons un peu avec ROS - turtlesim

Liste des services :

```
rosservice list
```

Infos sur le service set_pen :

```
rosservice info /turtle1/set_pen
rossrv show turtlesim/SetPen
```

Appeler le service set_pen :

```
rosservice call /turtle1/set_pen 10 10 10 25 0
(r= 10, g=10, b=10, width=25, off=0)
```

Jouons un peu avec ROS - turtlesim

Essayer les services ... :

/clear

/turtle1/teleport_absolute

/turtle1/teleport_relative

Jouons un peu avec ROS - turtlesim

Essayer les services ... :

/clear

→ **rosservice call /clear**

/turtle1/teleport_absolute

→ **rosservice call /turtle1/teleport_absolute 5.0 5.0 1.5**

/turtle1/teleport_relative

→ **rosservice call /turtle1/teleport_relative 2.0 1.5**

→ **rosservice call /turtle1/teleport_relative -- -2.0 -1.5**

La communauté ROS

- Les distributions : http://wiki.ros.org/Distributions
- Les packages : http://www.ros.org/browse/list.php
- ROS wiki : http://wiki.ros.org/
- ROS mailing list : http://discourse.ros.org/
- ROS answers : http://answers.ros.org/questions/