version 1.5

TIER IV ACADEMY
自動運転システム構築塾

Day2　ROS 演習

ROS演習１：catkin ビルドシステム

Tier IV
Intelligent Vehicle

この演習について

- ✓ ROS 演習 1 ：catkinビルドシステム
  - ROS 演習 2 ：ノードの作成とトピックの配信・購読
  - ROS 演習 3 ：TF の作成
  - ROS 演習 4 ：RViz での表示
  - ROS 演習 5 ：ROSBAG によるデータ記録

- ROS 演習 1 ～ 5 では ROS の基本的な要素技術を学習
- ROS 演習 1 ：catkin ビルドシステム
  - ROS のビルドシステムである catkin について学ぶ
  - 簡単な ROS のサンプルプログラムを書いて実行する

目次

ROS演習１：catkin ビルドシステム

第１章： ROS（Robot Operating Systems）

1.ROS とは

2.特徴

ROSとは

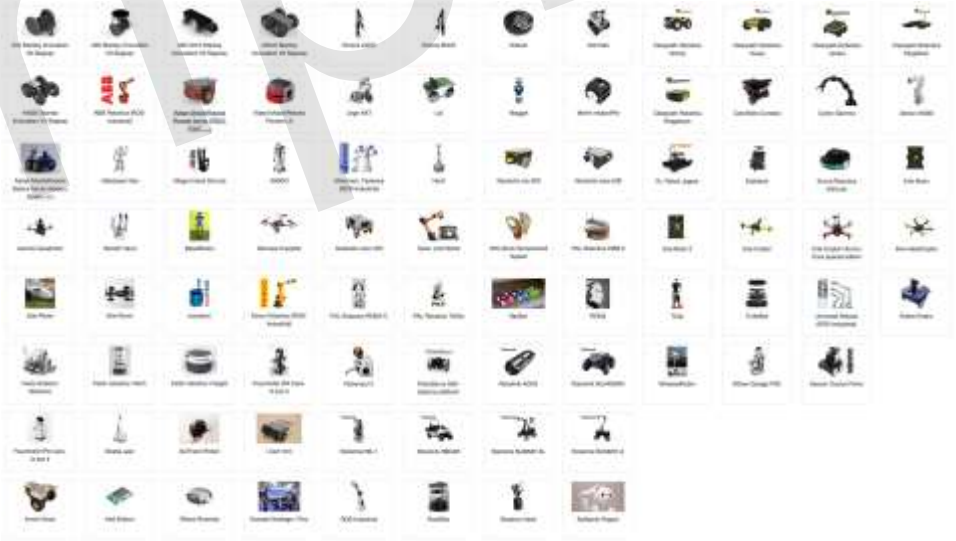
ROS (Robot Operating System)

: ロボット開発におけるライブラリやツールを提供

ハードウェアの抽象化, デバイスドライバ, ライブラリ, 視覚化ツール,
データ通信, パッケージ管理 ...etc

特長

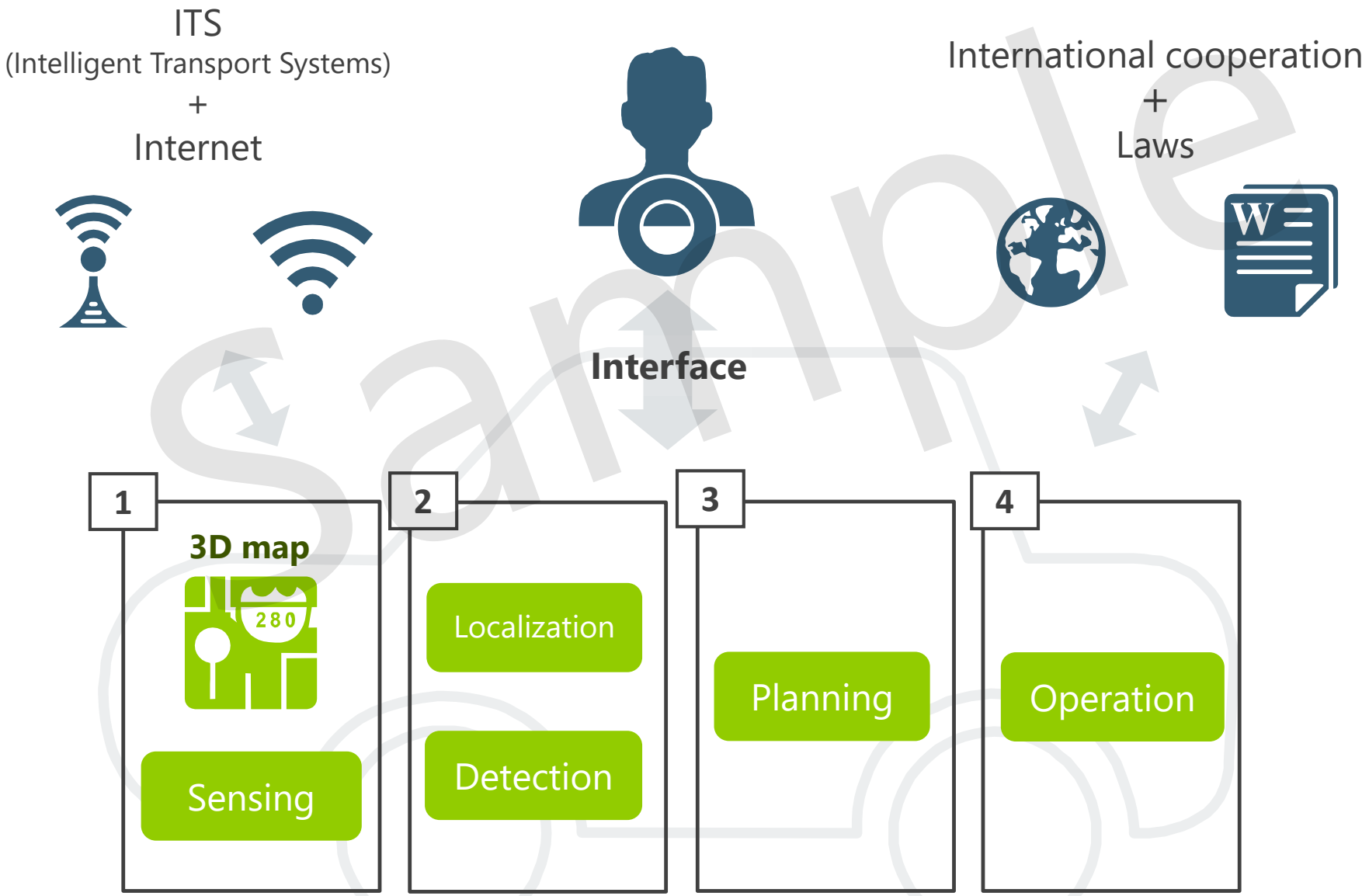
- 世界で最も利用されているロボットミドルウェア
- 豊富な対応ロボット・センサ
- オープンソース
- サポート言語： C++, Python
- 管理団体：OSRF
- 対応OS：Linux



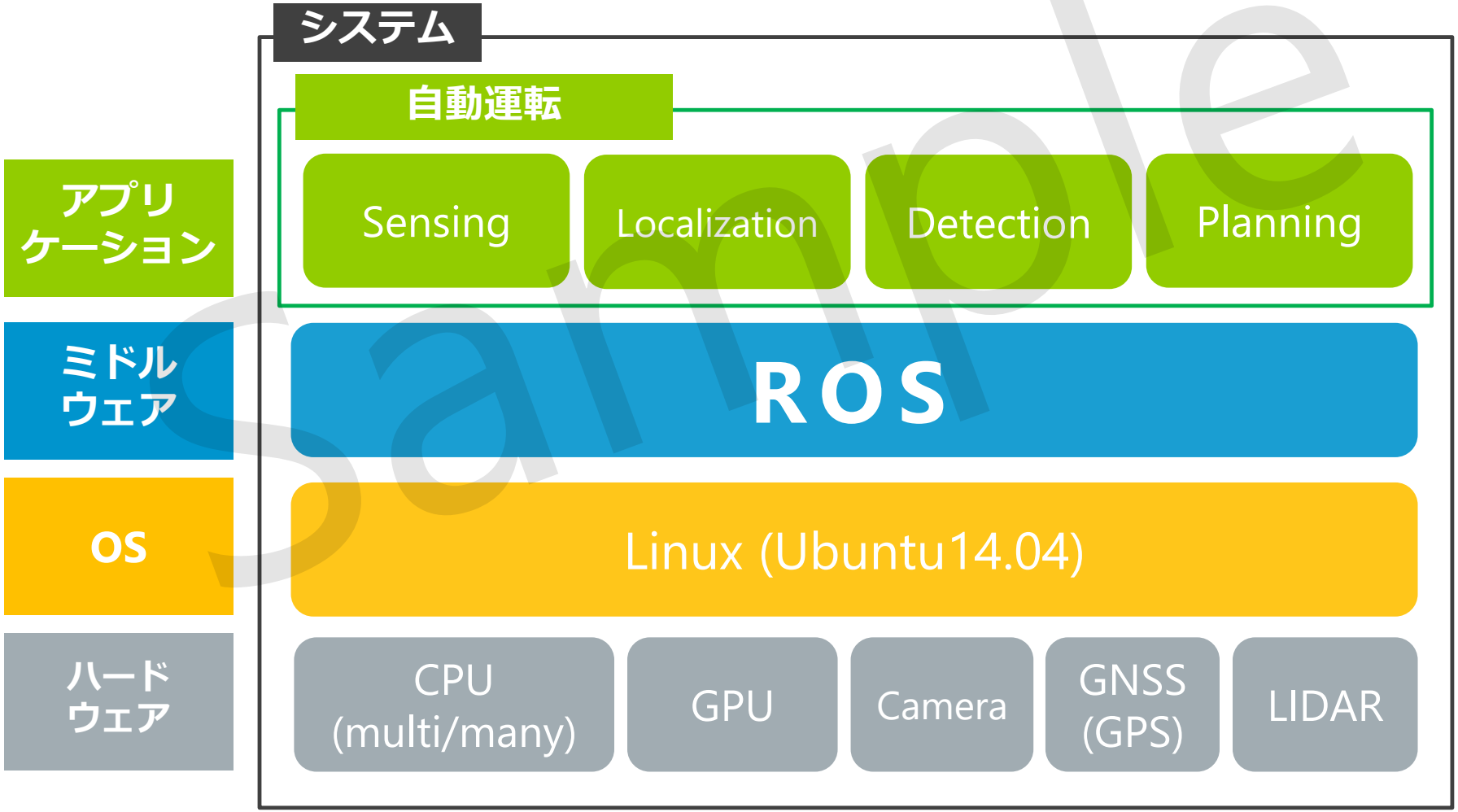
ubuntu

Open Source Robotics Foundation

自動運転システムにおけるROS (1/2)



自動運転システムにおけるROS (2/2)



ROS演習１：catkin ビルドシステム

第１章： ROS（Robot Operating Systems）

1.ROS とは

2.特徴

ROS の 特長

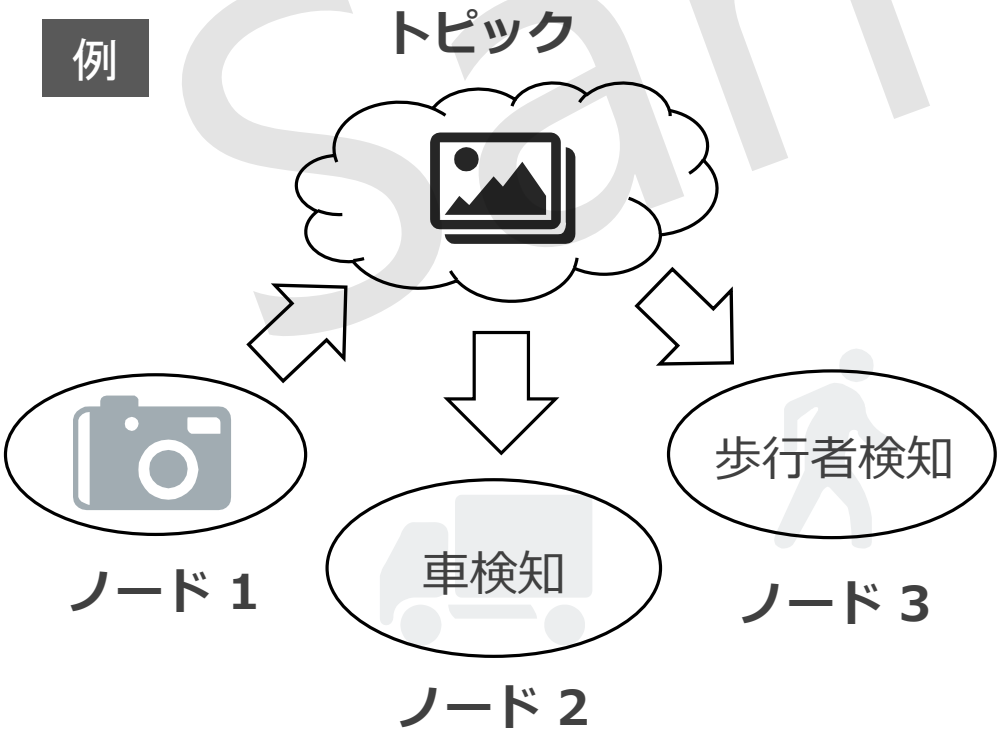
ROS (Robot Operating System)

：ロボット開発におけるライブラリやツールを提供

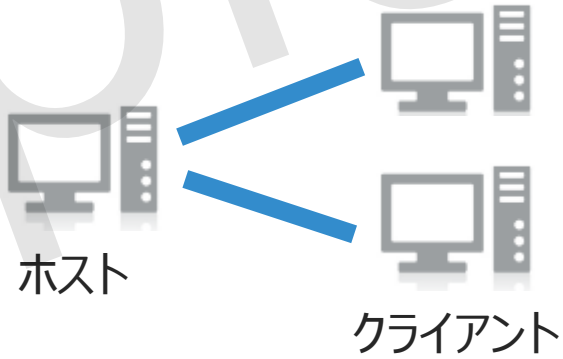
Publish / Subscribe モデル

- **ノード**の集合としてシステムを構築
- **トピック**を介してデータをやり取り

例



分散システム

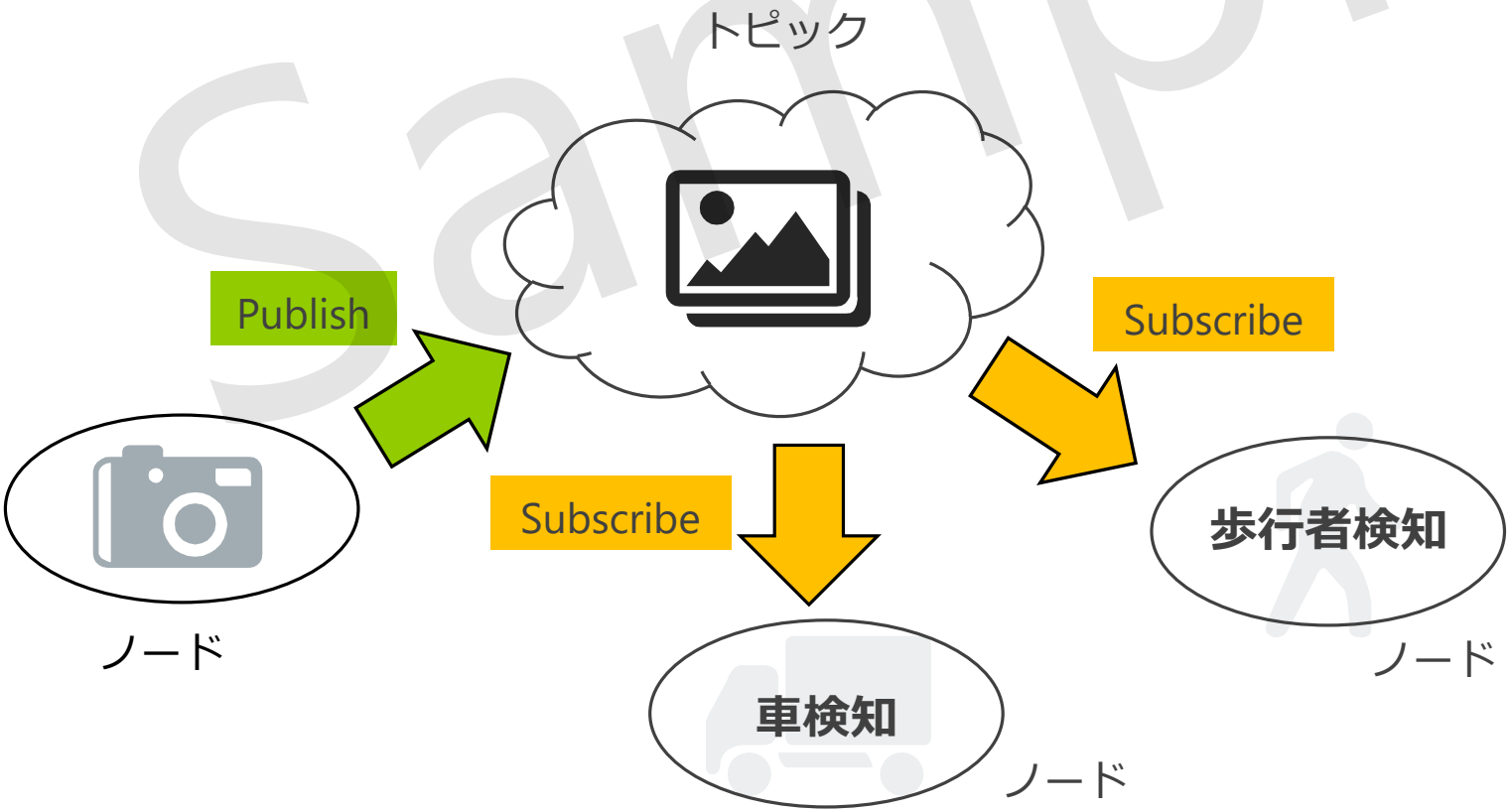


視覚化・シミュレーション

Publish / Subscribe モデル

処理を**ノード**として分割・管理し、**トピック**を介してデータのやり取りを行う。

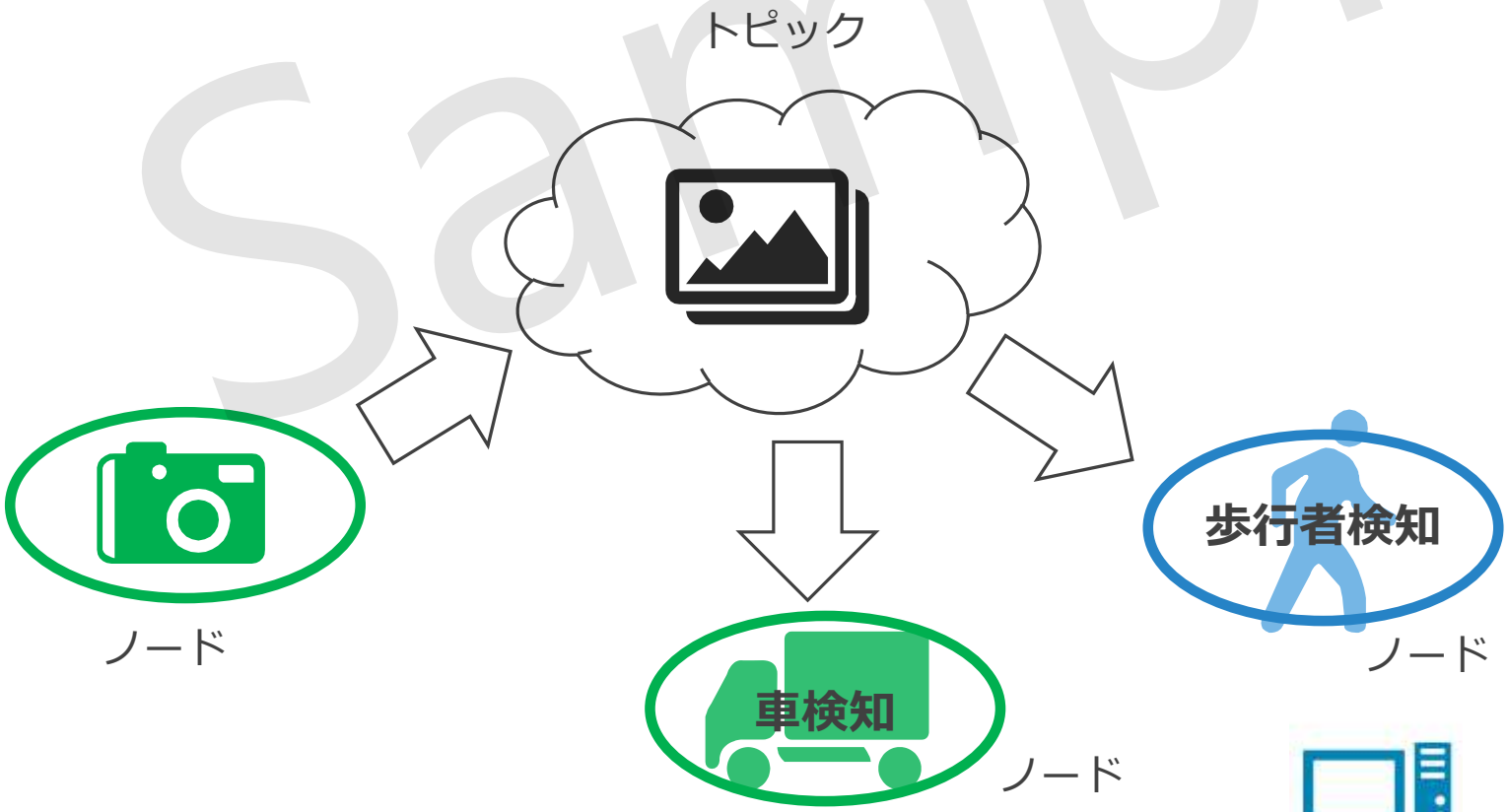
➡ 再利用性・生産性の向上, 分散環境への高い親和性, 障害分離



Publish / Subscribe モデル

処理を**ノード**として分割・管理し、**トピック**を介してデータのやり取りを行う。

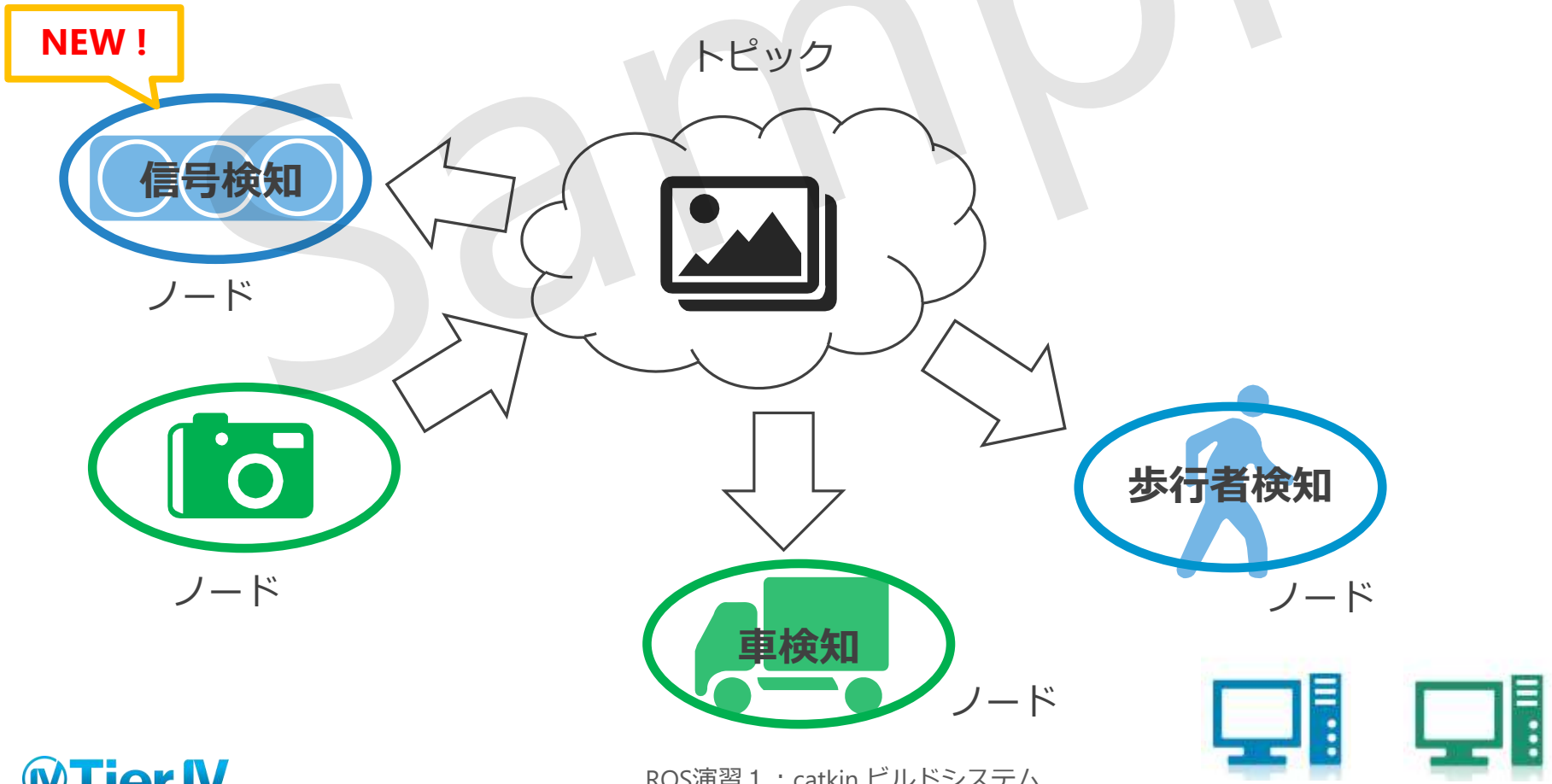
➡ 再利用性・生産性の向上, 分散環境への高い親和性, 障害分離



Publish / Subscribe モデル

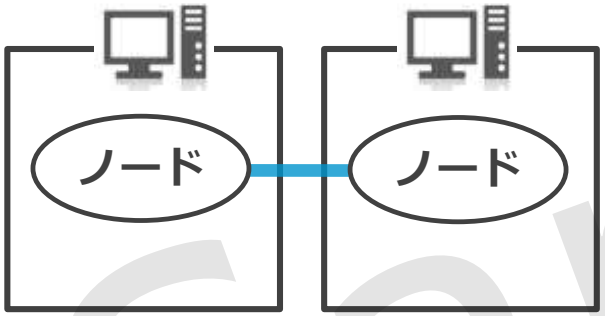
処理を**ノード**として分割・管理し、**トピック**を介してデータのやり取りを行う。

➡ 再利用性・生産性の向上, 分散環境への高い親和性, 障害分離

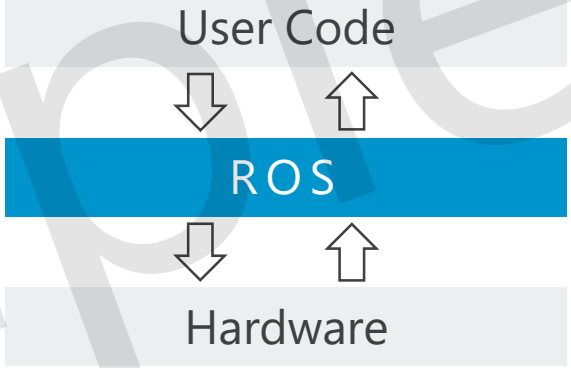


ROS の 特長

TCP/UDP or 共有メモリ



抽象化



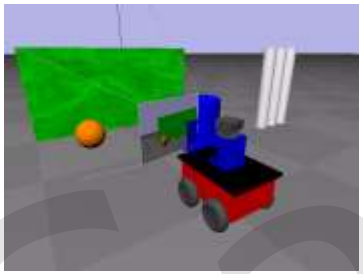
視覚化・シミュレーション

豊富なパッケージ
(デバイスドライバやライブラリ)

ROS の 特長

Gazebo：3D物理シミュレータ

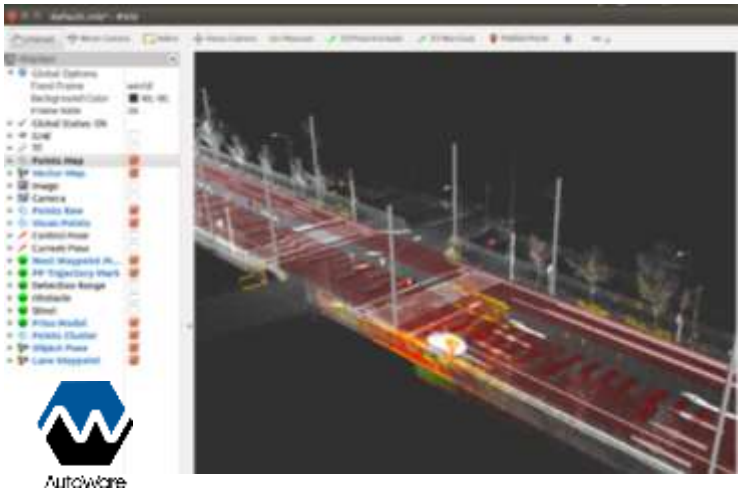
強力なGUIでROSとの連携が充実



RViz：3D視覚化ツール

簡単にシステム状態を視覚化可能

［再生データ］
- 記録したセンサデータ（**rosbag** ファイル）
- 指定した値のデータ



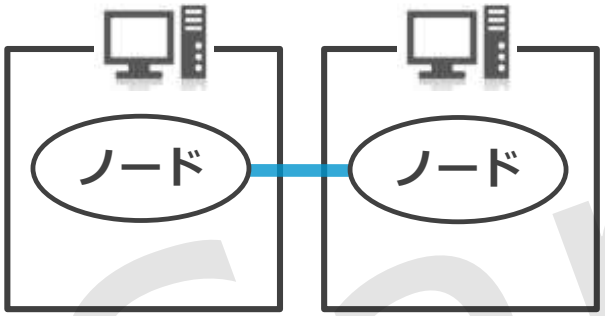
視覚化・シミュレーション

GAZEBO

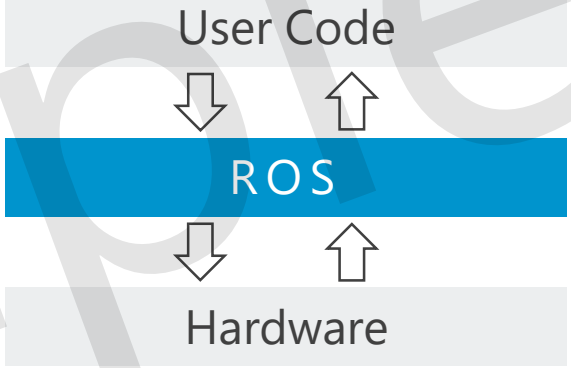
RViz

ROS の 特長

TCP/UDP or 共有メモリ



抽象化



視覚化・シミュレーション

豊富なパッケージ
(デバイスドライバやライブラリ)

ROS の 特長

ハードウェア

様々なロボットやセンサをサポート

パッケージ

2,000を超えるソフトウェアパッケージで効率的開発

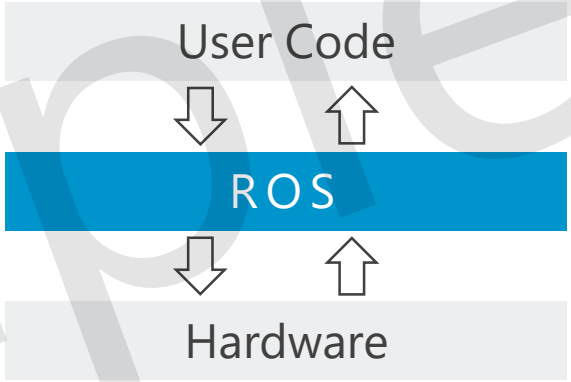
ライブラリ

座標変換・画像処理・点群処理など豊富にサポート

pcl

OpenCV

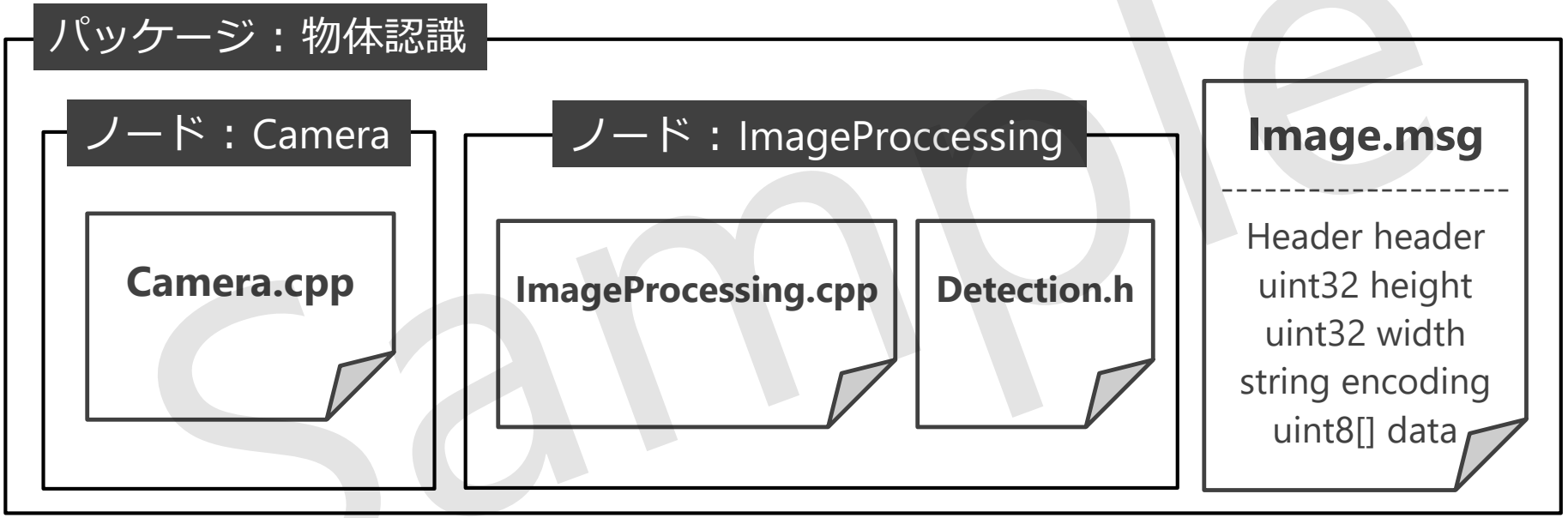
抽象化



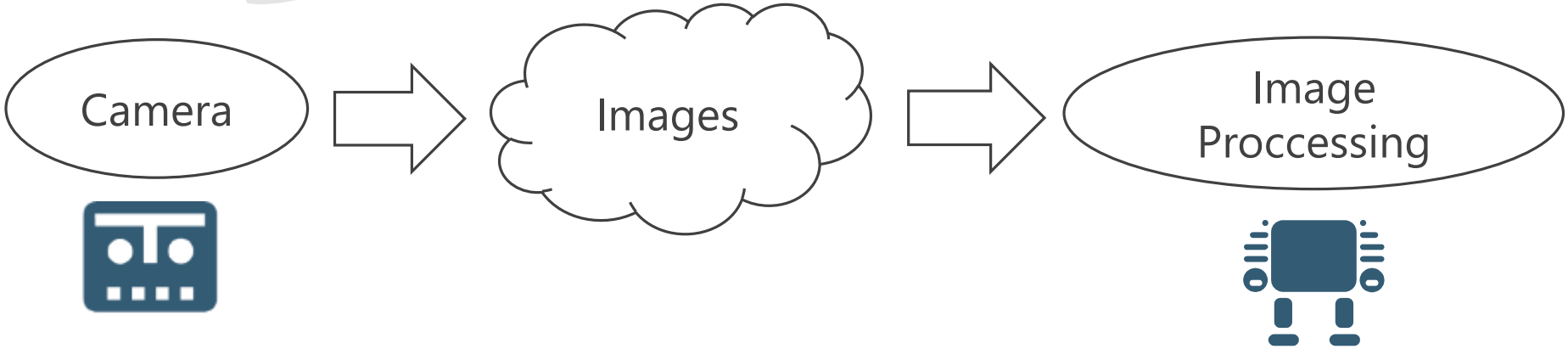
豊富なパッケージ
(デバイスドライバやライブラリ)

ROS の アプリケーション例

ファイル構成



システムモデル



ROS演習 1 ：catkin ビルドシステム

第 2 章 : ROS と catkin

ROS について (1/2)

ROS (Robot Operating System) とは

- ロボット向けのメタオペレーティングシステム
- ソフトウェア開発者のロボット・アプリ作成を支援
  - ✓ライブラリとツールを提供
    - ハードウェア抽象化, ライブラリ, 視覚化ツール, メッセージ通信, パッケージ管理など

- 良い点
  - ✓ 容易に理解・習得
  - ✓ オープンソース（無償）
  - ✓ 商用利用可能
  - ✓ 最先端の研究成果物が利用可能
  - ✓ ロボット分野のデファクトスタンダード
  - ✓ 最新のセンサー・アプリケーションが利用可能
  - ✓ コミュニティが活発
  - ✓ 公開パッケージが多数
  - ✓ ROS サポートのハードウェアが多数

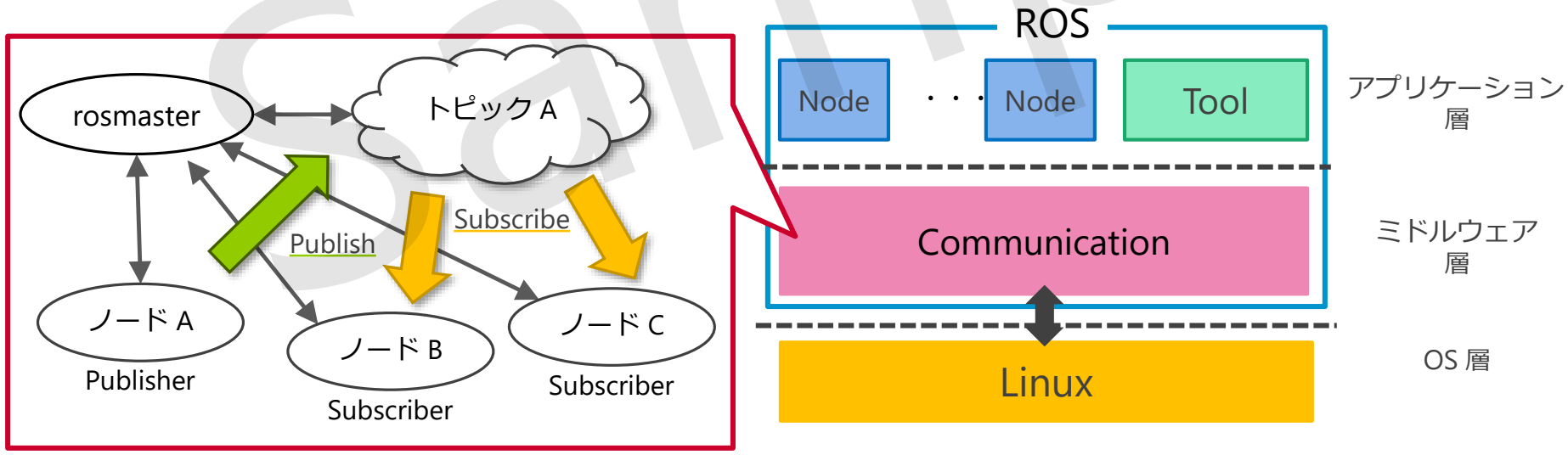
- 悪い点
  - ✓ 公式で Ubuntu のみの対応
  - ✓ リアルタイム性の検証が不十分

ROS について (2/2)

ROS の専門用語と通信モデル

トピック通信 ——— 基本となる通信方式

- ✓ Publish / Subscribe モデル
- ✓ ノード　：ソフトウェアの実行単位
- ✓ トピック ：ノード同士がメッセージのやり取りを行うための名前付きバス
- ✓ rosmaster：ノードやトピックの管理を行うノード



その他にもサービス、パラメータサーバーと言った通信方法があります（演習 3 にて）

catkin ビルドシステム

- ビルドシステムとは
  コンパイルやライブラリのリンクなどを行い、最終的な実行可能ファイルを作成を行うシステム
  - ✓ 例：Make, Cmake, Apache Ant, Autoconfなど

- catkin（キャッキン）とは
  CMake を拡張したROS のビルドシステム
  - ➢ CMake：様々なOS、コンパイラの差を吸収するビルドシステム
  - ➢ 前ROSのビルドシステムはrosbuild

  ROS のために開発されたが、ROS 専用のビルドシステムではない
  - ➢ ROS のソフトウェアシステムとは切り離されている
  - ➢ catkin では汎用的な CMake、make を利用しており、Eclipse などの統合開発環境でも利用可能

catkin を利用することで ROS に関連するビルドを効率的に行い
マルチプラットフォームで開発可能

ROSの新しいビルドシステムcatkinについて
http://myenigma.hatenablog.com/entry/20131229/1388320084



catkinの意味：
猫柳の花 or 子猫

第 2 章：演習

1. 環境構築
2. ソースコードの作成
3. ビルド
4. ROS ノードの実行

環境構築

- OSとROS のインストール（済）
- ROS が動作する標準的な環境を準備
  - PC：Intel系CPU（core-i3 以上を推奨)
  - OS：Ubuntu
  - 言語：C++, Python
  - ROS バージョン
    - ROS Indigo（2014 – 2019）
    - ROS Kinetic（2016 – 2021）
- 演習環境
  - OS：Ubuntu 14.04 LTS（Autoware 推奨）
  - ROS：Indigo（Autoware 推奨）

Ubuntu のインストール手順（済）
Ubuntu Japanese Teamサイト https://www.ubuntulinux.jp/
ROSの インストール手順（済）
ROS Wikiサイト http://wiki.ros.org/ja/indigo/Installation/Ubuntu
本演習で作成するファイルは下記の URL からもダウンロード可能
https://github.com/yukkysaito/TierIVAcademy

catkinワークスペースの作成 (1/3)

ワークスペース：以降プログラムを書く上で作業するディレクトリ

1. ROS コマンドを使えるようにするため、環境変数を設定

   $ source /opt/ros/indigo/setup.bash

   source ~/.bashrc に ”source /opt/ros/indigo/setup.bash“ を書き込むことで自動化可能

2. catkin ワークスペースを作成

   $ mkdir -p ~/catkin_ws/src

   catkin_ws：ワークスペース（名前は任意）
   src：ユーザ作成コードを格納するディレクトリ（名前は “src” 固定）

3. catkin_init_workspaceコマンドで、catkin ワークスペースを初期化

   $ cd ~/catkin_ws/src

   $ catkin_init_workspace

   初期化すると、以下のメッセージが出力される

```
Creating symlink "/home/iwasaki/catkin_ws/src/CMakeLists.txt" pointing to
  "/opt/ros/indigo/share/catkin/cmake/toplevel.cmake"
```

この時点で、トップレベルの CMakeLists.txt が作成される

catkinワークスペースの作成 (2/3)

4. ファイル構成を tree コマンドで確認

$ cd ~/catkin_ws/

$ tree

treeコマンドがインストールされていない場合
$ sudo apt-get install tree

5. catkin_makeコマンドで、catkin ワークスペースをビルド

$ cd ~/catkin_ws

$ catkin_make

catkinワークスペースの作成 (3/3)

6. ワークスペースを tree コマンドで確認

```
$ cd ~/catkin_ws
$ tree
```

build と devel の 2つのディレクトリが自動生成される

```
<~/>
 └ <catkin_ws/>
    ├ .catkin_workspace
    ├ <build/>
    │  ├ Makefile
    │  └ その他いろいろ
    ├ <devel/>
    │  ├ setup.bash
    │  └ その他いろいろ
    └ <src/>
       └ CMakeLists.txt - トップレベル
```

ビルドで自動生成される
各種の一時ファイルが格納

ビルドで生成された
ヘッダファイル・実行形式ファイルが格納

7. ビルドしたパッケージを読み込み

```
$ source ~/catkin_ws/devel/setup.bash
```

~/.bashrc の最終行に、コマンドを追加することで自動読み込み可能

ROS演習 1 ：catkin ビルドシステム

第 2 章：演習

1. 環境構築
2. ソースコードの作成
3. ビルド
4. ROS ノードの実行

パッケージの作成 (1/3)

パッケージの基本

- パッケージ：ROS コードのソフトウェア構成単位
- ディレクトリ単位で構成
  - パッケージの中にパッケージは作成不可
    - ✓ 複数のパッケージをまとめて扱うメタパッケージ という論理的な仕組みもある
  - パッケージディレクトリに直下に package.xml（マニフェスト）と CMakeLists.txt が存在
    - ✓ CMakeLists.txt：ビルド時に利用する設定ファイル
    - ✓ package.xml：パッケージの説明書

後に説明

個々のパッケージの基本ファイル構成

<my_package/>
├ <src/>
├ <include/>
├ CMakeLists.txt
└ package.xml

パッケージの作成 (2/3)

catkin ワークスペースにおけるパッケージの構成の例

```
<catkin_workspace/>
  ├ <build/> ...以下略
  ├ <devel/> ...以下略
  └ <src/>
      ├ CMakeLists.txt - トップレベル
      ├ <package_1/>
      │   ├ (いろいろ)
      │   ├ CMakeLists.txt
      │   └ package.xml
      │ ...略
      ├ <package_n/>
      │   └ ...略
      └ <subdirectory_A/>
          ├ <package_A1/>
          │ ...略
          └ <package_An/>
```

パッケージの作成 (3/3)

1. C++で書くパッケージを新規作成

   $ cd ~/catkin_ws/src

   $ catkin_create_pkg tutorials std_msgs roscpp

   ※catkin_create_pkg コマンドのフォーマット

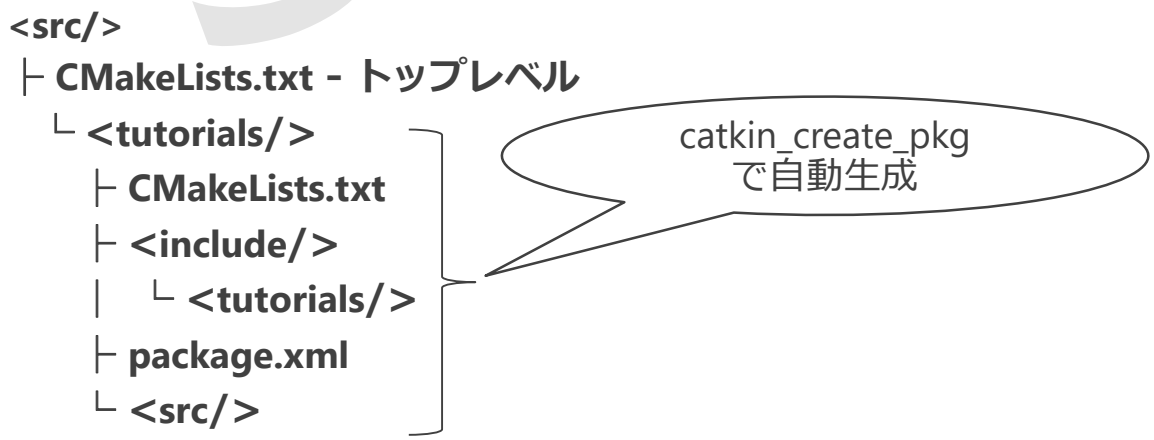
   catkin_create_pkg <package_name> [dependencies [dependencies ...]]

   - std_msgs：ROS の基本メッセージ定義（後で使用）
   - roscpp：ROS の基本 C++ライブラリ（※必須）

2. 作成したパッケージの内容を、treeコマンドで確認

   $ cd ~/catkin_ws/src

   $ tree



ソースコードの作成

最も簡単なROSのサンプルコードを作成

$ gedit ~/catkin_ws/src/tutorials/src/simple.cpp

以下の通りに simple.cpp のコードを入力。

```
#include "ros/ros.h" //ROS で必要なヘッダー
int main(int argc, char **argv)
{
  ros::init(argc, argv, "simple");  //ROS の初期化.ノードの名前を特定
  ros::NodeHandle n;  //プロセスのノードへのハンドラを作成
  ros::Rate loop_rate(1);  //1Hzで動作させるためのタイマーを作成
  int count = 0;
  while (ros::ok()) //Ctrl-C(SINGINTシグナル)を押すと ros::ok() が false を返却
    {
      ROS_INFO("Hello world %d", count++); //ログメッセージを出力
      loop_rate.sleep(); //1Hzで動作するようスリープ
    }
  return 0;
}
```

システム内では、
ユニークでなければならない
※実行時にリネーム可能

ROSの機能でログレベルに応じて表示可能
※ログレベルの強さ
ROS_DEBUG < ROS_INFO < ROS_WARN < ROS_ERROR < ROS_FATAL

第 2 章：演習

1. 環境構築
2. ソースコードの作成
3. ビルド
4. ROS ノードの実行

ビルドに必要な設定ファイル

ビルドに関する設定はCMakeLists.txtとpackage.xml

- CMakeLists.txt：CMakeで利用するビルド設定ファイル
  - CMake：コンパイラに依存しないビルド自動化ソフトウェアであり、様々なOSで動作
  - 参考： http://wiki.ros.org/ja/catkin/CMakeLists.txt

cmake_minimum_required：必要な CMake Version の指定
project()：Package Name
find_package()　：ビルドに必要な他の CMake/Catkin packages を見つける
catkin_package()：パッケージのビルドに必要な情報の指定
add_library()/add_executable()/target_link_libraries()：ライブラリや実行ファイルのビルド

- package.xml：パッケージの説明書であり、ビルドに必要なパッケージの依存関係などを記述
  - XMLと呼ばれる言語で記述
  - 参考：http://robotics.ait.kyushu-u.ac.jp/books/ROSBOOK_JP.pdf（P58, 59）

buildtool_depend：ビルドシステムの依存関係を記述
build_depend　　：パッケージをビルドするときに依存するパッケージ名を記述
run_depend　　：パッケージを実行するときに依存するパッケージ名を記述

CMakeLists.txtの修正

CMakeLists.txtをバックアップしてから修正

$ cd ~/catkin_ws/src/tutorials

$ cp CMakeLists.txt CMakeLists.org.txt

$ gedit CMakeLists.txt

- ## Build ##セクションに、コンパイルの指定を追加

  131: add_executable(simple src/simple.cpp)

  複数のソースファイルがある時は、空白区切りで列挙

- ## Build ##セクションに、リンクの指定を追加

  141: target_link_libraries(simple ${catkin_LIBRARIES})

  複数のリンクファイルがある時は、空白区切りで列挙

CMakeと同じ記法

package.xmlの修正

package.xmlをバックアップしてから修正

$ cd ~/catkin_ws/src/tutorials

$ cp package.xml package.org.xml

$ gedit package.xml

- メタ情報のメンテナンス担当者とライセンスを修正

  10: <maintainer email="メールアドレス">名前</maintainer>

  16: <license>BSD</license>

- それ以外のメタ情報については、任意記入

  もし追加の依存パッケージがあれば、ここで指定（今回はなし）

catkin_makeによるビルド

catkin_makeコマンドで、ワークスペース全体をビルド

$ cd ~/catkin_ws

$ catkin_make

ワークスペース内の特定のパッケージのみをビルドする。

$ catkin_make --pkg tutorials

ビルドされた実行ファイルは、以下にある。

~/catkin_ws/devel/lib/tutorials/simple

第 2 章：演習

1. 環境構築
2. ソースコードの作成
3. ビルド
4. ROS ノードの実行

ROSノードの実行 (1/2)

- 別端末(以下同)で、roscoreコマンド（rosmaster）を起動

  $ roscore

  ROSを利用した分散システムではどれか１つのマシンでroscoreを起動

  参考： http://robot.isc.chubu.ac.jp/?p=538

- rosnodeコマンドで、現在起動しているROSのノードを確認のために表示

  $ rosnode list

  /rosout ← roscoreを起動すると現れるノード

- rosrunコマンドを使って、simpleをノードとして実行

  参考：コマンド形式：rosrun <package> <executable>

  $ rosrun tutorials simple

  [INFO] [XXXXXXXXX.XXXXXXXXX]: Hello world 0

  [INFO] [XXXXXXXXX.XXXXXXXXX]: Hello world 1

  :

- simple 実行ファイルは、直接実行することも可能

  $ ~/catkin_ws/devel/lib/tutorials/simple

ROSノードの実行 (2/2)

- rosnodeコマンドで、現在起動しているROSのノードを確認するために表示

  $ rosnode list

  /rosout

  /simple

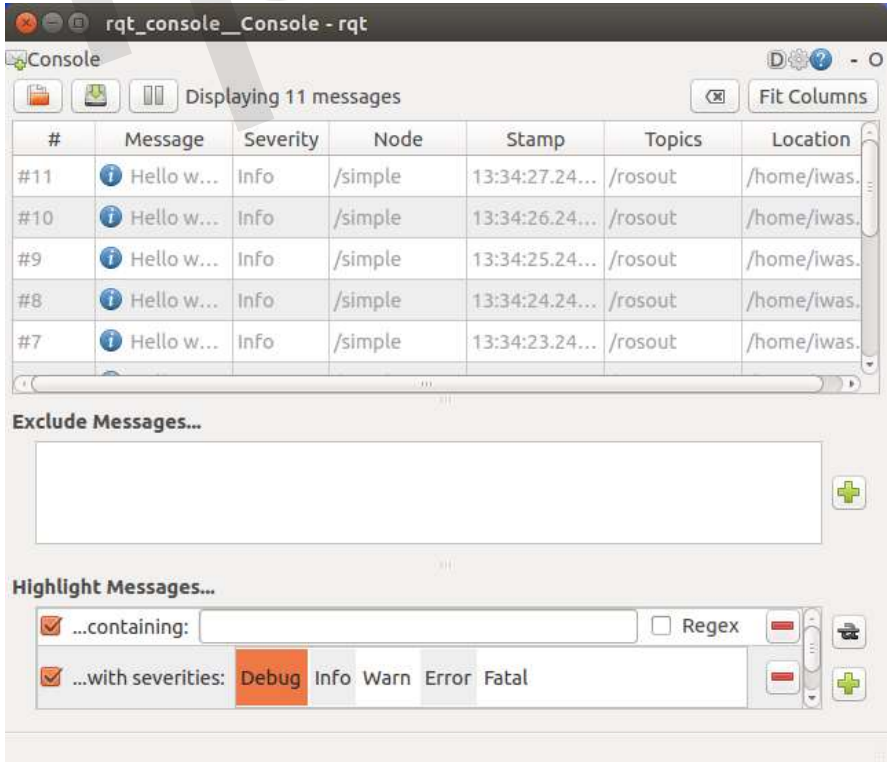
- rqt_consoleコマンドで、ROS_INFOから出力されるメッセージを見る

  $ rqt_console

- Ctrl-Cで、全ての端末の実行を終了

  参考：ノードを終了させるには、
  以下のコマンドを実行（推奨）。

  $ rosnode kill simple



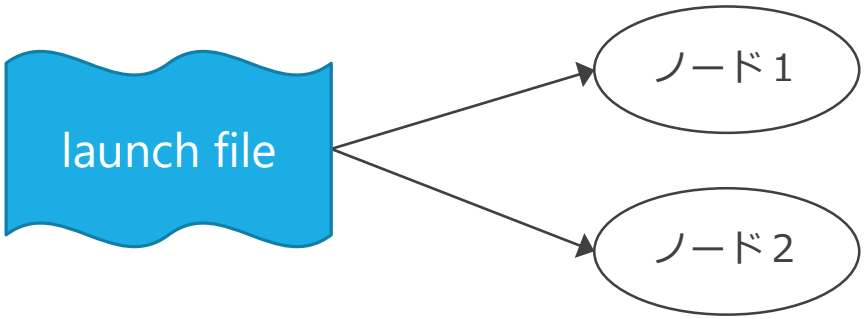
roslaunchとは

- 幾つものノードを立ち上げる際には手間

  Autowareでは数十～ノードを立ち上げる必要がある。

AutowareではGUIで
ボタンクリックにより、
roslaunchが起動する仕組み

- roslaunch：パッケージ内の複数のノードを一度に起動
  - その他、実行時に変更可能パラメータなども記述可能
    - ✓ノード名のリネーム
    - ✓トピック名の変更
    - ✓パラメータの設定
  - launch file：XML形式で書かれたスクリプト



roslaunchによる一つのROSノードの実行

1. パッケージ内に、launch ディレクトリを作成

   $ mkdir ~/catkin_ws/src/tutorials/launch

2. launch ディレクトリ内に、simple.launchファイルを作成

   $ gedit ~/catkin_ws/src/tutorials/launch/simple.launch

```
<launch>
  <node pkg="tutorials" name="simple" type="simple" output="screen"/>
</launch>
```

3. roslaunch コマンドで roscore と simpleノードを起動

   ※コマンド形式：roslaunch [package_name] [filename.launch]

   $ roslaunch tutorials simple.launch

4. rosnode コマンドで、現在起動しているROSのノードを表示

   $ rosnode list

   /rosout

   /simple

5. Ctrl-Cで、全ての端末の実行を終了

roslaunchによるノード名のリネーム

1. simple.launchファイルを編集

   $ gedit ~/catkin_ws/src/tutorials/launch/simple2.launch

```
<launch>
  <node pkg="tutorials" name="simple2" type="simple" output="screen"/>
</launch>
```

クラスとインスタンスのように扱われる

2. roslaunchコマンドで roscore と simpleノードを起動

   ※コマンド形式：roslaunch [package_name] [filename.launch]

   $ roslaunch tutorials simple2.launch

3. rosnodeコマンドで、現在起動しているROSのノードを表示

   $ rosnode list

   /rosout

   /simple2

4. Ctrl-Cで、全ての端末の実行を終了

roslaunchによるnamespaceとROSノードの実行 (1/2)

namespace：別々のnamespaceで区切ることで同一のノード名が存在可能

1. launchディレクトリ内に、simple_ns.launchファイルを作成

   $ gedit ~/catkin_ws/src/tutorials/launch/simple_ns.launch

2. simple_ns.launchファイルの内容として、以下を入力

ns は namespace の略

```
<launch>
 <group ns="simple1">
  <node pkg="tutorials" name="simple" type="simple" output="screen"/>
 </group>
 <group ns="simple2">
  <node pkg="tutorials" name="simple" type="simple" output="screen"/>
 </group>
</launch>
```

roslaunchによるnamespaceとROSノードの実行 (2/2)

3. roslaunchコマンドで実行

   $ roslaunch tutorials simple_ns.launch

4. rosnodeコマンドで、現在起動しているROSのノードを表示

   $ rosnode list

   /rosout

   /simple1/simple

   /simple2/simple

5. rqt_consoleコマンドで2つのノードからのログ出力を確認

   $ rqt_console

6. Ctrl-Cで、全ての端末の実行を終了

Tier IV

Intelligent Vehicle

www.tier4.jp

Appendix

- Linux の利用
- 参考文献
- 関連リンク１
- 関連リンク２

Linux コマンド一覧

コマンド	機能	使用例
cd [dir]	指定したディレクトリ(フォルダ)へ移動する	**$ cd ~/Autoware/ros/src**
cp [source] [dest]	指定したファイルやディレクトリをコピーする	**$ cp CMakeLists.txt CMakeLists.org.txt**
gedit [file]	指定したファイルを gedit エディタで開く	**$ gedit CMakeLists.txt**
ls	ディレクトリの情報を表示する	**$ ls**
mkdir [dir]	指定したディレクトリを作成する	**$ mkdir ~/catkin_ws**
mkdir -p [dir/-/dir]	指定したディレクトリをサブディレクトリごと作成する	**$mkdir –p ~/catkin_ws/src**
source [file]	指定した設定ファイルを読み込む	**$ source /opt/ros/indigo/setup.bash**
tree	ファイルやディレクトリの構成を木構造で表示する	**$ tree**
./[file]	指定したファイルを実行する	**$ ./run**

ROS コマンド一覧

コマンド	機能	使用例
catkin_init_workspace	catkin ワークスペースを作成する	**$ catkin_init_workspace**
catkin_make	catkin ワークスペースをビルドする	**$ catkin_make**
catkin_make -pkg [package]	catkin ワークスペース内の特定のパッケージのみをビルドする	**$ catkin_make -pkg tutorials**
roscore	rosmaster を起動する	**$ roscore**
roslaunch [package] [.launch]	roscore と launch ファイルで設定したノードを起動する	**$ roslaunch tutorials simple.launch**
rosmsg show [message type]	指定したメッセージがもつデータの種類を調べる	**$ rosmsg show geometry_msg/Twist**
rosnode kill [node name]	指定したノードを終了させる	**$ rosnode kill rosout**
rosnode list	起動しているノードの一覧を表示する	**$ rosnode list**
rosrun [package] [node name]	指定したノードを実行する	**$ rosrun tutorials simple**
rostopic echo [topic name]	指定したトピックがもつメッセージの内容を表示する	**$ rostopic echo /turtle1/cmd_vel**
rostopic type [topic name]	指定したトピックのメッセージ型を表示する	**$ rostopic type /turtle1/cmd_vel**

端末でのディレクトリ移動

1
```
academy31@academy31:~$ cd
.autoware/              .mozilla/               Documents/
.cache/                 .nv/                    Downloads/
.compiz/                .opencv_install/        Music/
.config/                .pip/                   NVIDIA_CUDA-8.0_Samples/
.cupy/                  .pki/                   Pictures/
.dbus/                  .ros/                   Public/
.emacs.d/               .rviz/                  Templates/
.gconf/                 ACC/                    Videos/
.local/                 Autoware/               catkin_ws/
.mozc/                  Desktop/                tools/
```
2
```
academy31@academy31:~$ cd A
ACC/      Autoware/
```
3
```
academy31@academy31:~$ cd Autoware/
```

1. **$ cd** と入力した状態でTabを押すと、現在のディレクトリ内のフォルダが表示される
2. **$ cd A** と入力した状態でTabを押すと、その後に続くことが可能なものだけが表示される
3. **$ cd Au**と入力した状態でTabを押すと、残りが補完される（Autowareと入力される）

```
academy31@academy31:~$ cd Autoware/ros/src
academy31@academy31:~/Autoware/ros/src$
```

現在の位置

一度に複数ディレクトリを移動することも可能

```
academy31@academy31:~/Autoware/ros/src$ cd ../
academy31@academy31:~/Autoware/ros$
```

一つ上のディレクトリに移動

$ cd ../

```
academy31@academy31:~/Autoware/ros$ cd ~/catkin_ws
academy31@academy31:~/catkin_ws$
```

どのディレクトリからでもhomeからの移動が可能

$ cd ~/

注意事項

- gedit や roscore を立ち上げた端末ではプロセスを終了させるまでは他のコマンドを使うことはできない
  - ✓ コマンドを打っても反映されない場合は $ があるか確認をする
  - ✓ 他のコマンドを利用する場合には、新しい端末を作成する
  - ✓ 同一の端末で利用したい場合は、後ろに & をいれる

  例）$ gedit CMakeLists.txt &

- コマンドの後ろにはスペースを入れる

  誤）$ mkdir–p~/catkin_ws/src

  正）$ mkdir –p ~/catkin_ws/src

- Tab を使ってタイピングミスを減らす
  - ✓ 端末上で Tab キーを打つことで、タイピング可能な文字列の表示、及び文字列の補完を行う

- Ctrl + C でノードを切ることができる

参考文献

ROS に関する本

- 『詳説 ROSロボットプログラミング（導入からSLAM・Gazebo・MoveItまで）』
  表 允晢・倉爪 亮・渡邊 裕太(2015)　ISBN 9784990873608
  http://robotics.ait.kyushu-u.ac.jp/books/ROSBOOK_JP.pdf
  （無償公開）

- 『ROSではじめるロボットプログラミング　-フリーのロボット用「フレームワーク」』
  小倉 崇(2015)　ISBN 4777519015
  https://www.amazon.co.jp/dp/4777519015/ref=cm_sw_r_tw_dp_x_Zct-xbKQFVEYE

- ROSプログラミング
  銭 飛(2016)　ISBN 4627853416
  https://www.amazon.co.jp/dp/4627853416/ref=cm_sw_r_tw_dp_x_sct-xbVGBXKH6

参考リンク 1

- ROS についての情報・勉強サイト
  - ROS開発者向けWikiサイト
    http://wiki.ros.org/　英語サイト
    http://wiki.ros.org/ja/ 日本語翻訳サイト（他11カ国語翻訳）
  - ROSチュートリアル
    http://wiki.ros.org/ja/ROS/Tutorials (日本語翻訳サイト)

演習では、ROS Wikiにあるチュートリアルをベースにしています

参考リンク 2

- Ubuntu のインストール手順
  - Ubuntu Japanese Teamサイト
    https://www.ubuntulinux.jp/

- ROS のインストール手順
  - ROS Wiki サイト
    http://wiki.ros.org/ja/indigo/Installation/Ubuntu

- catkin について
  - ROSの新しいビルドシステムcatkinについて
    http://myenigma.hatenablog.com/entry/20131229/1388320084