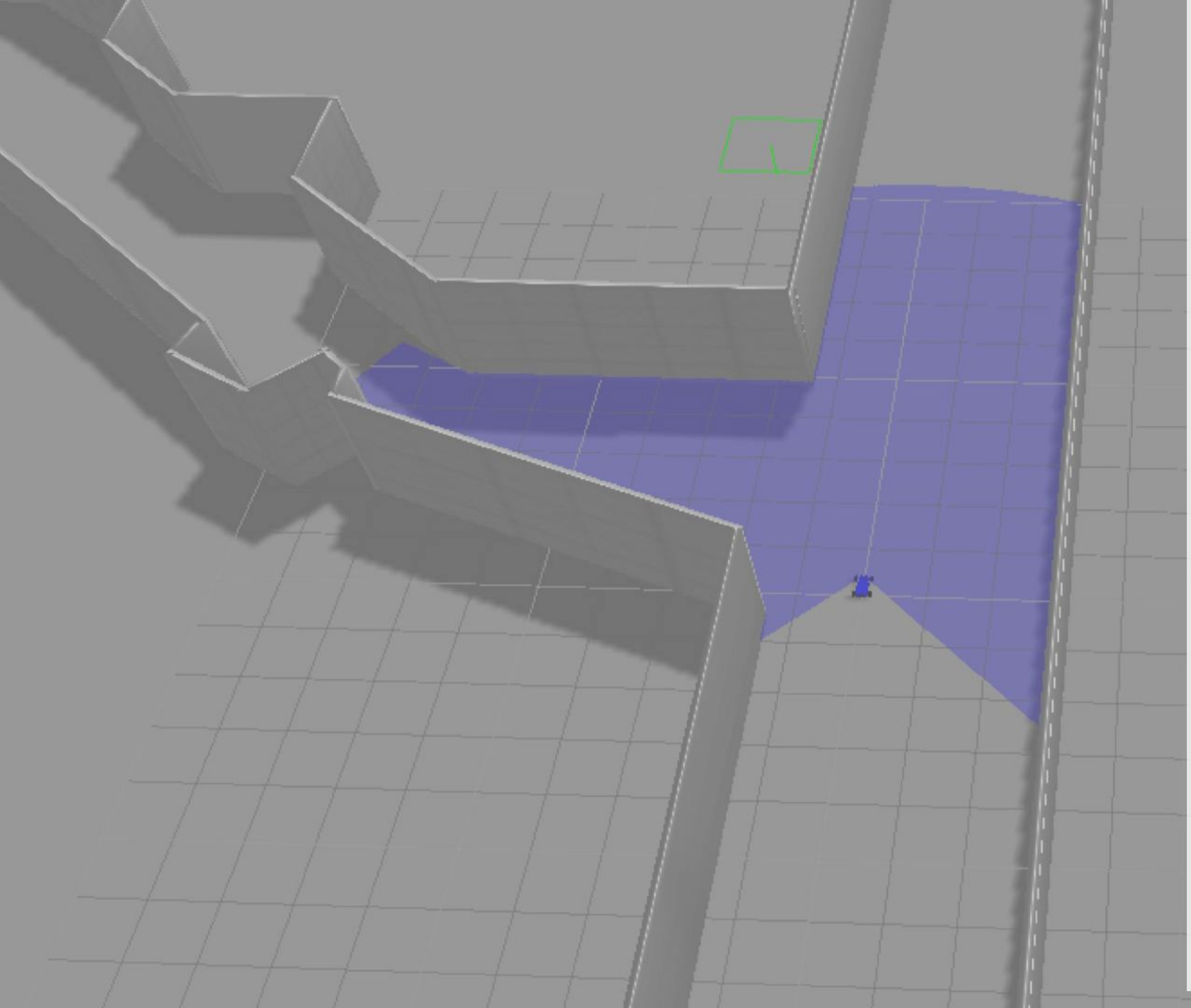


F1/10
Autonomous Racing

ROS
Filesystem and Workspaces

Madhur Behl

CS 4501/SYS 4582
Spring 2019
Rice Hall 120

Announcements

- Assignment 1 due on Wednesday, Feb 6 at 2pm. (before the lab)

- Office hours:

  - **Varundev Sukhil**
    - Mo and We
    - 10– 11am
    - Link Lab 204

  - Madhur Behl
    - Thu 1-2pm
    - Link Lab 265

ROS Nodes

- Single-purpose, executable program
- Individually compiled, executed, and managed
- Organized in *packages*

Run a node with

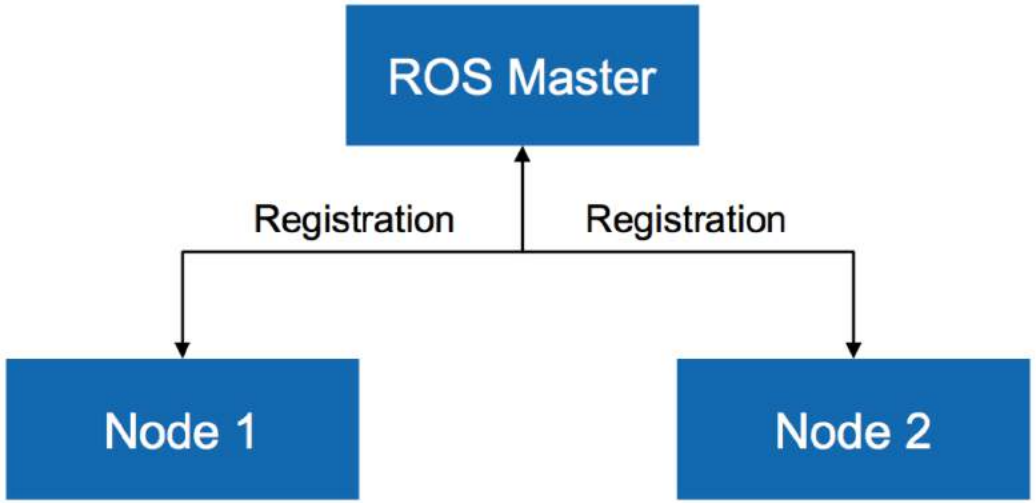
```
> rosrun package_name node_name
```

See active nodes with

```
> rosnode list
```

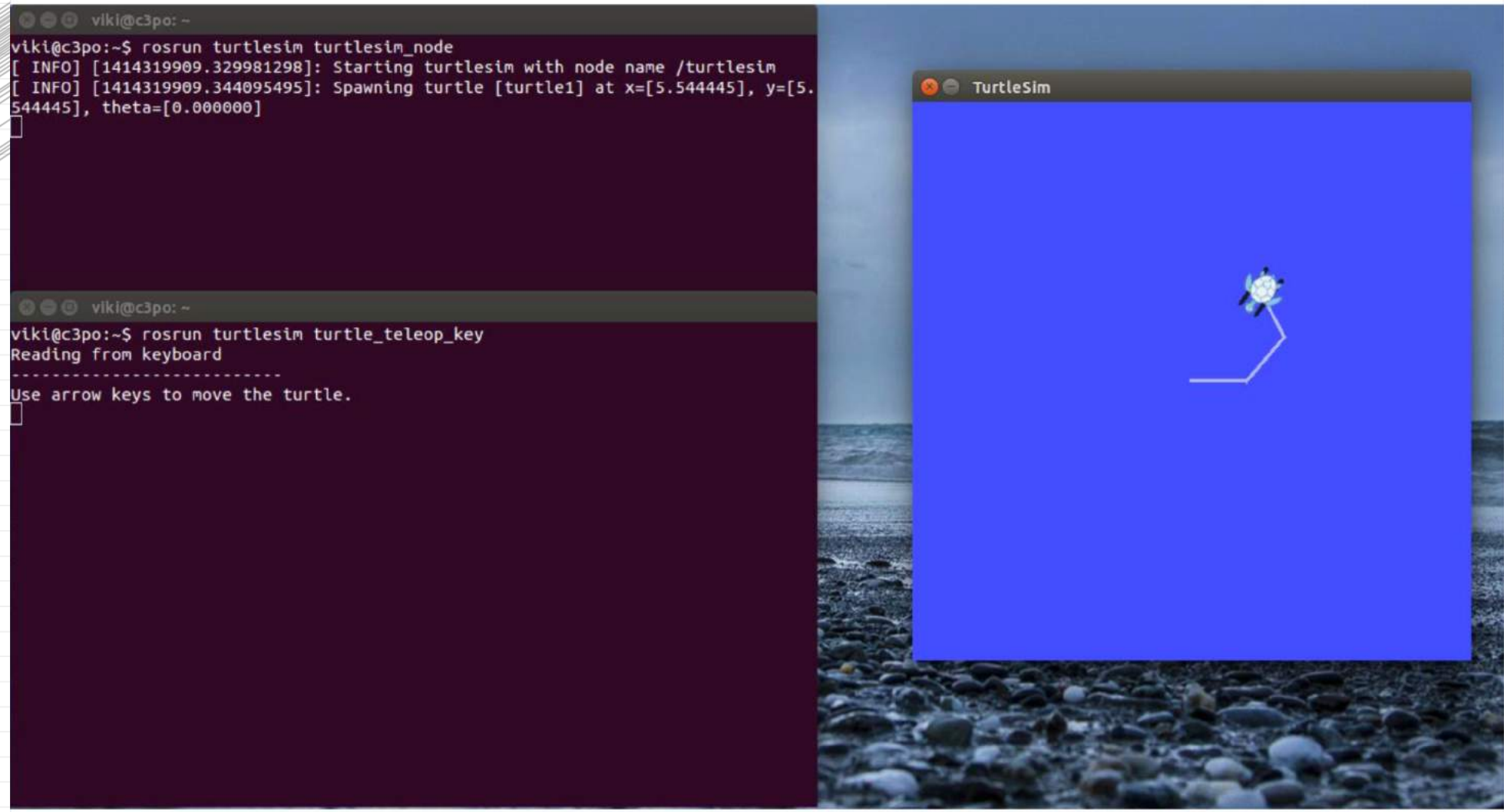
Retrieve information about a node with

```
> rosnode info node_name
```



More info
http://wiki.ros.org/rosnode

- **Demo – Turtlesim:** In separate terminals, run:
  - roscore
  - rosrun turtlesim turtlesim_node
  - rosrun turtlesim turtle_teleop_key



rosnode info

```
viki@c3po: ~
viki@c3po:~$ rosnode info turtlesim
--------------------------------------------------------------------------------
Node [/turtlesim]
Publications:
 * /turtle1/color_sensor [turtlesim/Color]
 * /rosout [rosgraph_msgs/Log]
 * /turtle1/pose [turtlesim/Pose]

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

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


contacting node http://c3po:54205/ ...
Pid: 3825
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound
    * transport: TCPROS
 * topic: /turtle1/cmd_vel
    * to: /teleop_turtle (http://c3po:47526/)
    * direction: inbound
    * transport: TCPROS

viki@c3po:~$
```

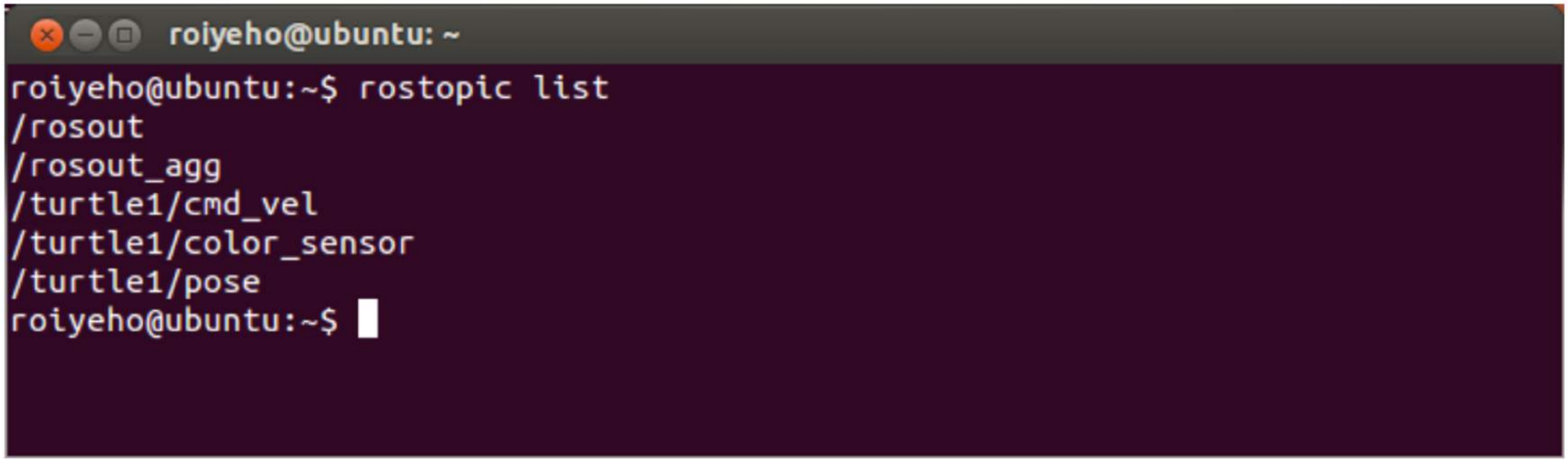
rostopic

- Gives information about a topic and allows to publish messages on a topic

Command	
$rostopic list	List active topics
$rosnode echo /topic	Prints messages of the topic to the screen
$rostopic info /topic	Print information about a topic
$rostopic type /topic	Prints the type of messages the topic publishes
$rostopic pub /topic type args	Publishes data to a topic

rostopic list

- Displays the list of current topics:



Publish to ROS Topic

- Use the **rostopic pub** command to publish messages to a topic
- For example, to make the turtle move forward at a 0.2m/s speed, you can publish a cmd_vel message to the topic /turtle1/cmd_vel:

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist '{linear: {x: 0.2, y: 0, z: 0}, angular: {x: 0, y: 0, z: 0}}'
```

- To specify only the linear x velocity:

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist '{linear: {x: 0.2}}'
```

Topic Type Arguments

Proper way to terminate ROS nodes

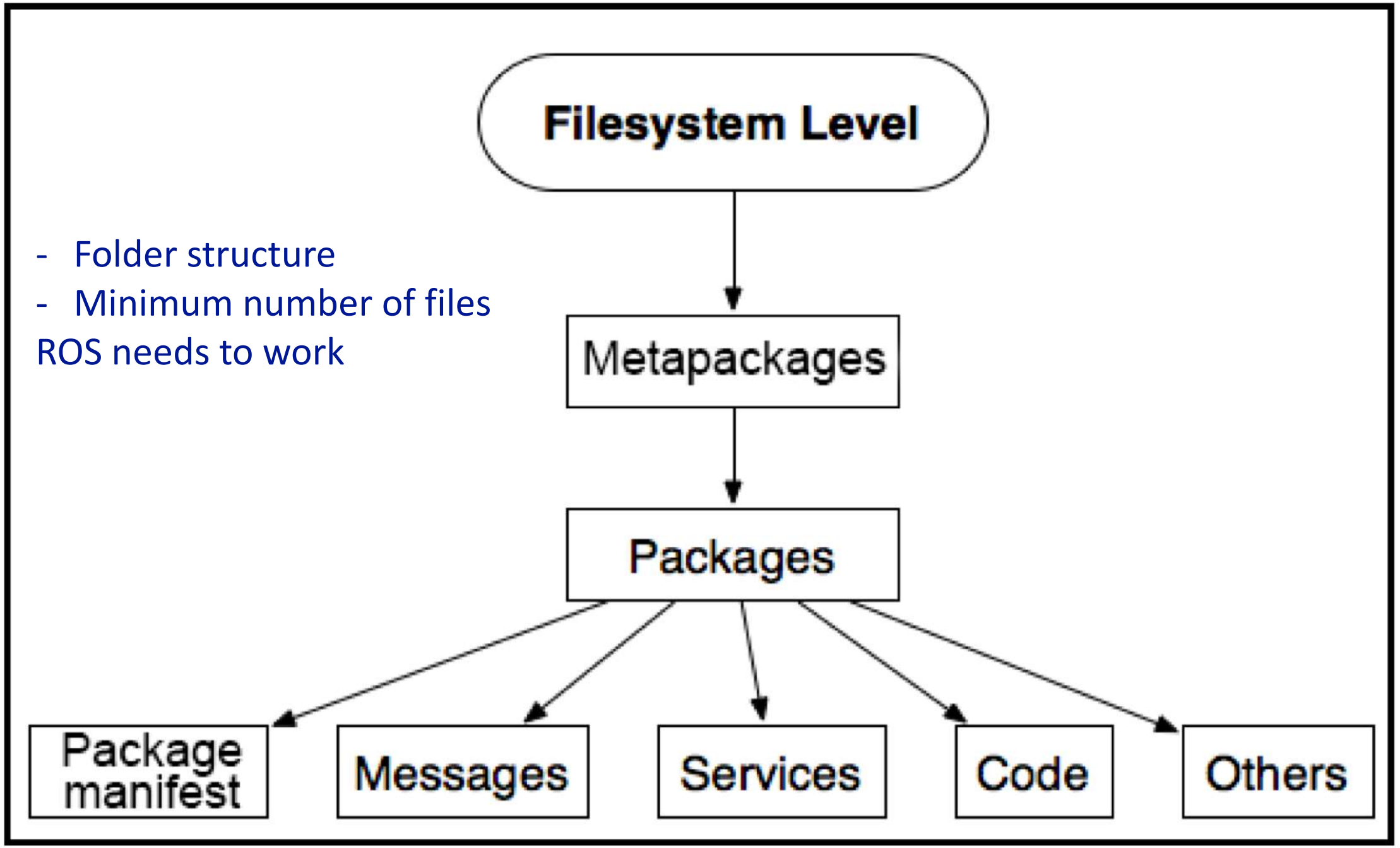
rosnode kill *node_name*

You can also kill a node using the usual Ctrl-C *technique. However, that method may not give the node a chance to unregister itself from the master. A symptom of this problem is that the killed node may still be listed by* `rosnode list` *for a while. This is harmless, but might make it more difficult to tell what's going on. To remove dead nodes from the list, you can use this command:*

```
rosnode cleanup
```

- Folder structure
- Minimum number of files

ROS needs to work



The main goal of the ROS Filesystem is to centralize the build process of a project, while at the same time provide enough flexibility and tooling to decentralize its dependencies.

catkin Build System

- **catkin** is the ROS build system
  - The set of tools that ROS uses to generate executable programs, libraries and interfaces
- The original ROS build system was **rosbuild**
  - Still used for older packages (before 2015)
- Implemented as custom CMake macros along with some Python code

Just FYI: CMake is an open-source, cross-platform family of tools designed to build, test and package software.

rosbuild

- The former ROS build system
  - But still used by many packages
- Collection of custom scripts + CMake + make
- Suffered from some limitations
  - Mostly impacting ROS packagers
    - Those tasked with turning ROS source code into redistributable binaries
    - Portability, standards compliance, cross-compiling, ease of packaging
  - Worked quite well for researchers / rapid prototyping
- Should not be used for new ROS modules
- http://www.ros.org/wiki/rosbuild

catkin

- The current ROS build system
  - Official as of ROS Groovy
- Goals:
  - Reuse existing tools (CMake macros)
  - Be as standard compliant as possible
  - Facilitate binary packaging
    - "install" target
- Modules using catkin are called "wet"
  - Packages still using rosbuild are called "dry"
  - Analogy of a rising water line
    - rosbuild packages can depend on catkin packages, but not the other way around
    - ROS modules are converted to catkin from the bottom of the dependency chain to the top

Catkin: Slim, cylindrical flower cluster from a willow tree

ROS Workspaces

- Folders for modifying, building, executing one or more source packages
  - With special directory layout understood by ROS tools
- Packages in workspace are often related by function or project
  - For example: all source packages needed for the FALCONS project
  - Also reasonable: all the source packages that I maintain
- Workspace packages are built with one command
  - Usually faster that building each package individually
- User may have multiple workspaces
  - e.g. a workspace for each project, temp workspaces for quick changes
- Typically, only one workspace is “active”
- Activate a workspace set by sourcing it’s *setup.bash* script
  - Must be sourced in each terminal window
  - Behind the scenes: sets ROS environment variables
- Active workspace “overlays” installed packages
  - That is, you can modify, build, run versions of installed packages

catkin Build System

The catkin workspace contains the following spaces

Work here

src

The *source space* contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.

Don't touch

build

The *build space* is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.

Don't touch

devel

The *development (devel) space* is where built targets are placed (prior to being installed).

If necessary, clean the entire build and devel space with

```
> catkin clean
```

More info

http://wiki.ros.org/catkin/workspaces

Creating a catkin Workspace

- Initially, the workspace will contain only the top-level CMakeLists.txt

```
$ mkdir ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

- **catkin_make** command builds the workspace and all the packages within it

```
cd ~/catkin_ws
catkin_make
```

catkin Workspace

- A set of directories in which a set of related ROS code lives
- You can have multiple ROS workspaces, but you can only work in one of them at any one time
- Contains the following spaces:

Source space	Contains the source code of catkin packages. Each folder within the source space contains one or more catkin packages.
Build Space	is where CMake is invoked to build the catkin packages in the source space. CMake and catkin keep their cache information and other intermediate files here.
Development (Devel) Space	is where built targets are placed prior to being installed
Install Space	Once targets are built, they can be installed into the install space by invoking the install target.

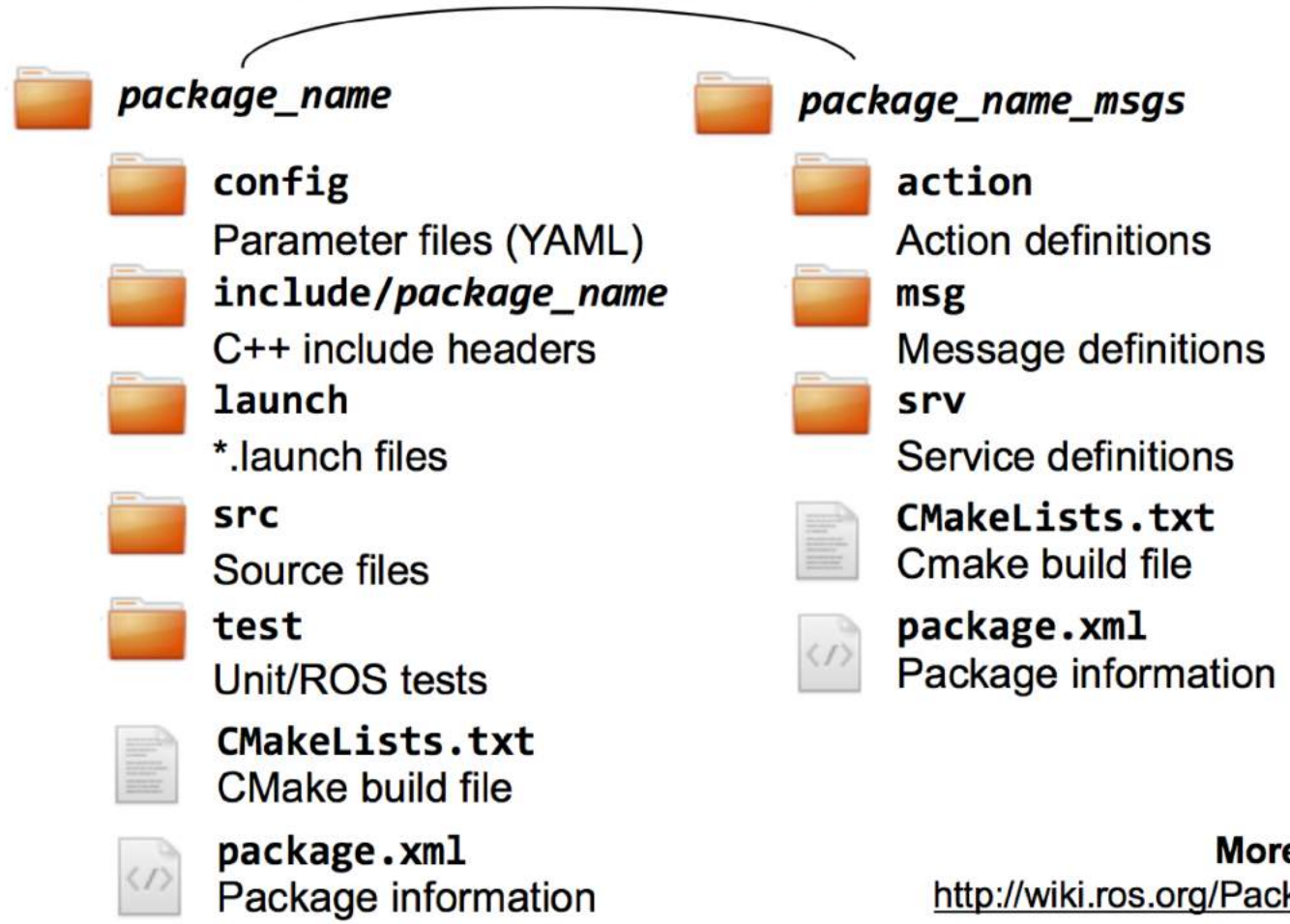
ROS Packages

- ROS software is organized into *packages*, which can contain source code, launch files, configuration files, message definitions, data, and documentation
- A package that builds up on/requires other packages (e.g. message definitions), declares these as *dependencies*

To create a new package, use

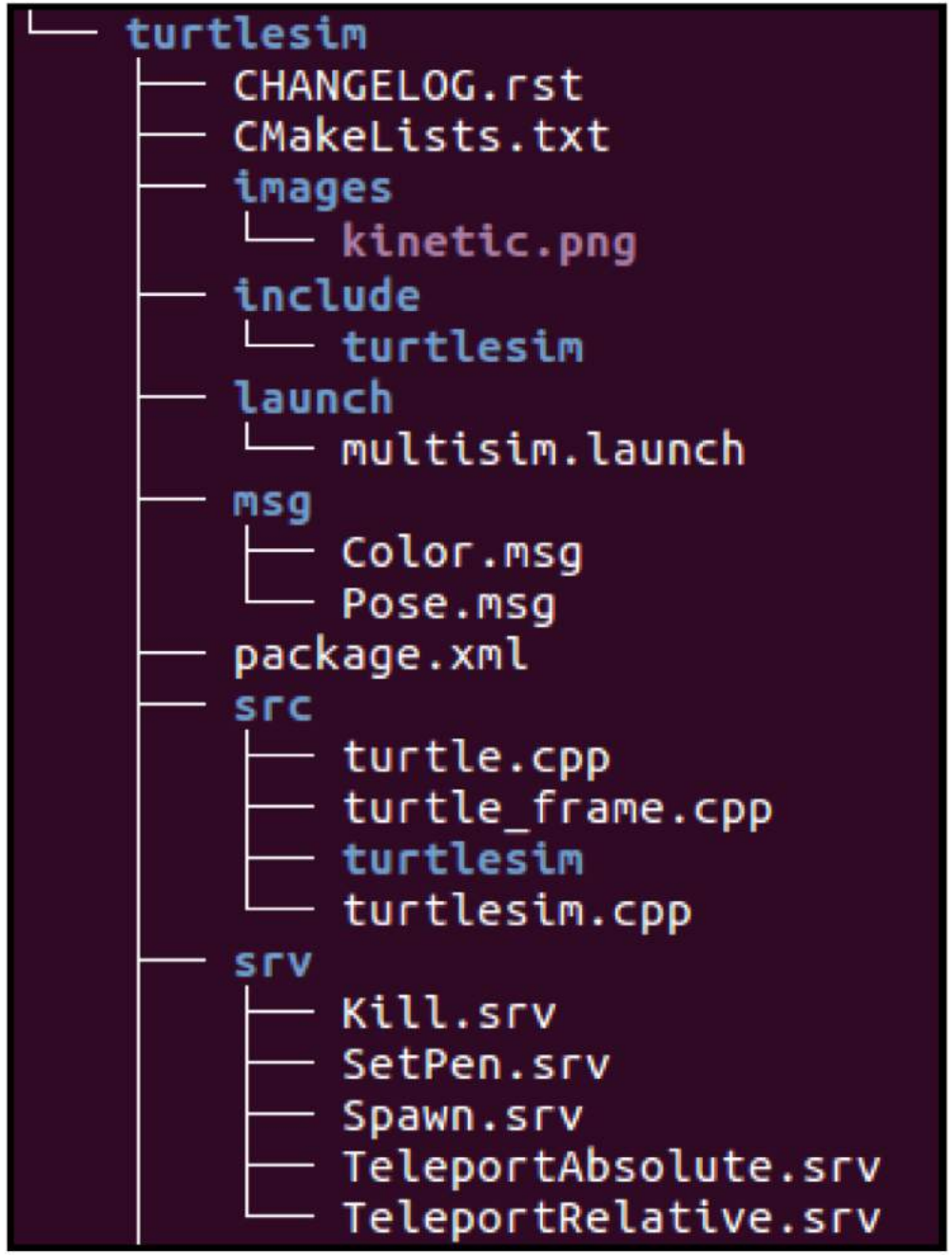
```
> catkin_create_pkg package_name
    {dependencies}
```

Separate message definition packages from other packages!



More info
http://wiki.ros.org/Packages

Turtlesim Package



ROS packages

- To create a new package, navigate to your workspace and then use the **catkin_create_pkg** utility.

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg beginner_tutorials std_msgs
rospy roscpp
```

- To build the new created package do:

```
$ cd ~/catkin_ws
$ catkin_make
$ . ~/catkin_ws/devel/setup.bash
$ rospack profile
```

Packages organization in the ROS workspace

```
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
```

ROS Packages
package.xml

Must be included with any catkin-compliant package's root folder.

- The `package.xml` file defines the properties of the package
  - Package name
  - Version number
  - Authors
  - **Dependencies on other packages**
  - …

package.xml

```
<?xml version="1.0"?>
<package format="2">
  <name>ros_package_template</name>
  <version>0.1.0</version>
  <description>A template for ROS packages.</description>
  <maintainer email="pfankhauser@e…">Peter Fankhauser</maintainer>
  <license>BSD</license>
  <url type="website">https://github.com/ethz-asl/ros_best_pr…</url>
  <author email="pfankhauser@ethz.ch">Peter Fankhauser</author>

  <buildtool_depend>catkin</buildtool_depend>

  <depend>roscpp</depend>
  <depend>sensor_msgs</depend>
</package>
```

More info
http://wiki.ros.org/catkin/package.xml

ROS Packages
CMakeLists. txt

The `CMakeLists.txt` is the input to the CMakebuild system

1. Required CMake Version (`cmake_minimum_required`)
2. Package Name (`project()`)
3. Find other CMake/Catkin packages needed for build (`find_package()`)
4. Message/Service/Action Generators (`add_message_files()`, `add_service_files()`, `add_action_files()`)
5. Invoke message/service/action generation (`generate_messages()`)
6. Specify package build info export (`catkin_package()`)
7. Libraries/Executables to build (`add_library()`/`add_executable()`/`target_link_libraries()`)
8. Tests to build (`catkin_add_gtest()`)
9. Install rules (`install()`)

CMakeLists.txt

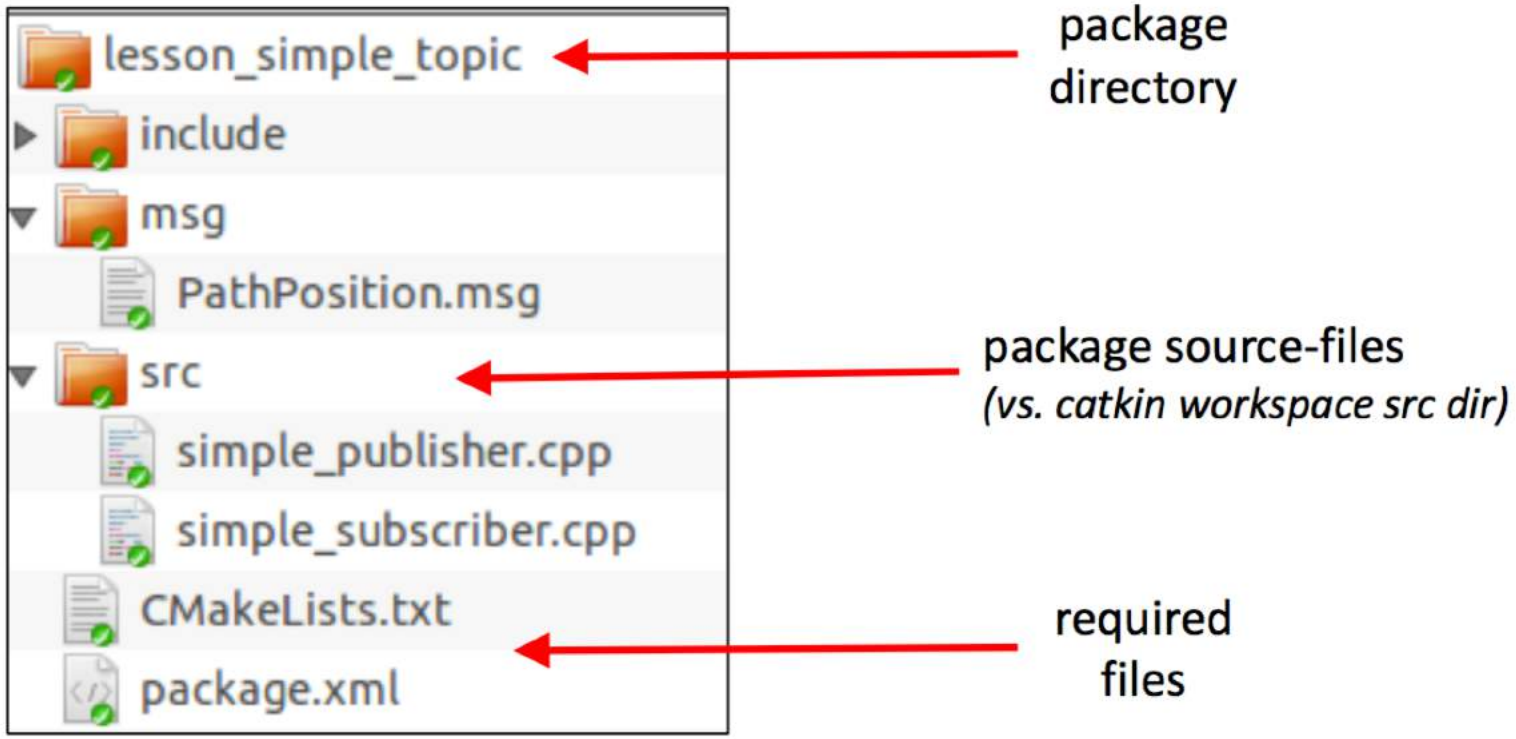
```
cmake_minimum_required(VERSION 2.8.3)
project(ros_package_template)

## Use C++11
add_definitions(--std=c++11)

## Find catkin macros and libraries
find_package(catkin REQUIRED
  COMPONENTS
    roscpp
    sensor_msgs
)

…
```

More info
http://wiki.ros.org/catkin/CMakeLists.txt

Common Files and Directories

Directory	Explanation
include/	C++ include headers
src/	Source files
msg/	Folder containing Message (msg) types
srv/	Folder containing Service (srv) types
launch/	Folder containing launch files
package.xml	The package manifest
CMakeLists.txt	CMake build file

- ROS components are organized into **packages**
- Packages contain several **required files**:
  - `package.xml`
    - **metadata** for ROS: package name, description, dependencies, ...
  - `CMakeLists.txt`
    - **build rules** for catkin



Where are the Packages?

- **Installed packages: *`/opt/ros/indigo`***
  - **Package parts divided into *lib*, *include*, and *share* subdirectories**
    - **Filesystem Hierarchy Standard (FHS)**
  - **Owned by root**
    - **Look, but don't touch**
  - **Typically does not include the source code (except for Python)**
- **User packages: anywhere**
  - **But usually in user's home directory**

Anywhere? How does ROS find the packages?

Navigating Across ROS Packages

- Users typically have many ROS packages in many places
- Frequently need to:
  - Jump from one package to another
  - Reference a file in another package
- rosbash suite – command line tools for quick package navigation & use
  - `roscd` - change directory to a package name
    - Example: `roscd robot_spinner/src`
  - `rospd` - pushd equivalent of roscd
  - `rosd` - lists directories in the directory-stack
  - `rosls` - list files of a ros package
  - `rosed` - edit a file in a package
  - `roscp` - copy a file from a package
  - `rosrun` - run executables of a ros package
- `rospack` – get detailed information about a a package
  - Examples:
    ```
    rospack find robot_spinner
    rospack depends robot_spinner
    ```
  - Replicate roscd functionality: `cd `rospack find robot_spinner`/src`

package.xml – package meta information supporting documentation, compilation, and distribution

- Required: name, description, version, maintainer(s), license
- Optional: authors, url's, dependencies, plugins, …

CMakeLists.txt – the main CMake file used to build the package

- Standard and catkin-specific CMake functions / macros
  - "Reads" the package.xml file
  - Finds other catkin packages to access libraries / include directories
  - Export items for other packages depending on you

Check package dependencies

```
$ rospack depends1 beginner_tutorials
```

```
roscpp
rospy
std_msgs
```

package.xml : Description tag

First update the description tag:

Toggle line numbers

```
5    <description>The beginner_tutorials package</description>
```

Change the description to anything you like, but by convention the first sentence should be short while covering the scope of the package. If it is hard to describe the package in a single sentence then it might need to be broken up.

package.xml : Maintainer tag

Next comes the maintainer tag:

Toggle line numbers

```
7   <!-- One maintainer tag required, multiple allowed, one person per tag -->
8   <!-- Example:  -->
9   <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
10  <maintainer email="user@todo.todo">user</maintainer>
```

This is a required and important tag for the package.xml because it lets others know who to contact about the package. At least one maintainer is required, but you can have many if you like. The name of the maintainer goes into the body of the tag, but there is also an email attribute that should be filled out:

Toggle line numbers

```
7   <maintainer email="you@yourdomain.tld">Your Name</maintainer>
```

package.xml : license tag

Next is the license tag, which is also required:

Toggle line numbers

```
12   <!-- One license tag required, multiple allowed, one license per tag -->
13   <!-- Commonly used license strings: -->
14   <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
15   <license>TODO</license>
```

You should choose a license and fill it in here. Some common open source licenses are BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, and LGPLv3. You can read about several of these at the Open Source Initiative. For this tutorial we'll use the BSD license because the rest of the core ROS components use it already:

Toggle line numbers

```
8   <license>BSD</license>
```

package.xml : Build dependencies

Toggle line numbers

```
27   <!-- The *_depend tags are used to specify dependencies -->
28   <!-- Dependencies can be catkin packages or system dependencies -->
29   <!-- Examples: -->
30   <!-- Use build_depend for packages you need at compile time: -->
31   <!--   <build_depend>genmsg</build_depend> -->
32   <!-- Use buildtool_depend for build tool packages: -->
33   <!--   <buildtool_depend>catkin</buildtool_depend> -->
34   <!-- Use exec_depend for packages you need at runtime: -->
35   <!--   <exec_depend>python-yaml</exec_depend> -->
36   <!-- Use test_depend for packages you need only for testing: -->
37   <!--   <test_depend>gtest</test_depend> -->
38   <buildtool_depend>catkin</buildtool_depend>
39   <build_depend>roscpp</build_depend>
40   <build_depend>rospy</build_depend>
41   <build_depend>std_msgs</build_depend>
```

package.xml : Execution dependencies

Toggle line numbers

```
12    <buildtool_depend>catkin</buildtool_depend>
13
14    <build_depend>roscpp</build_depend>
15    <build_depend>rospy</build_depend>
16    <build_depend>std_msgs</build_depend>
17
18    <exec_depend>roscpp</exec_depend>
19    <exec_depend>rospy</exec_depend>
20    <exec_depend>std_msgs</exec_depend>
```

Toggle line numbers

```
 1 <?xml version="1.0"?>
 2 <package format="2">
 3   <name>beginner_tutorials</name>
 4   <version>0.1.0</version>
 5   <description>The beginner_tutorials package</description>
 6
 7   <maintainer email="you@yourdomain.tld">Your Name</maintainer>
 8   <license>BSD</license>
 9   <url type="website">http://wiki.ros.org/beginner_tutorials</url>
10   <author email="you@yourdomain.tld">Jane Doe</author>
11
12   <buildtool_depend>catkin</buildtool_depend>
13
14   <build_depend>roscpp</build_depend>
15   <build_depend>rospy</build_depend>
16   <build_depend>std_msgs</build_depend>
17
18   <exec_depend>roscpp</exec_depend>
19   <exec_depend>rospy</exec_depend>
20   <exec_depend>std_msgs</exec_depend>
21
22 </package>
```

package.xml Example

```
<?xml version="1.0"?>
<package>
    <name>robot_spinner</name>
    <version>0.2.12</version>
    <description>Motion planner to spin a robot around</description>
    <maintainer email="boulet@ll.mit.edu">Michael Boulet</maintainer>
    <license>BSD</license>

    <url type="website">http://www.ros.org/wiki/robot_spinner</url>
    <author>Michael Boulet</author>

    <buildtool_depend>catkin</buildtool_depend>

    <build_depend>roscpp</build_depend>
    <build_depend>geometry_msgs</build_depend>

    <run_depend>roscpp</run_depend>
    <run_depend>geometry_msgs</run_depend>

    <export>
        <nodelet plugin="${prefix}/robot_spinner_nodelet.xml"/>
    </export>
</package>
```

Should match the directory name

Tools used on building platform needed to build this package

Dependencies needed to build this package

Dependencies needed to run this package / dependencies other packages need to build against or use this package (often the same as `build_depend`)

Container for "extra information" subsystems or other packages need to embed

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(robot_spinner)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS roscpp std_msgs)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)


## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
# catkin_python_setup()

######################################
## Declare ROS messages and services ##
######################################

## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )

## Generate services in the 'srv' folder
# add_service_files(
#   FILES
#   Service1.srv
#   Service2.srv
# )

## Generate added messages and services with any dependencies listed here
# generate_messages(
#   DEPENDENCIES
#   std_msgs
# )

###################################
## catkin specific configuration ##
###################################
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if you package contains header files
## LIBRARIES: libraries you create in this project that dependent projects also need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also need
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES robot_spinner
#  CATKIN_DEPENDS roscpp std_msgs
#  DEPENDS system_lib
)

###########
## Build ##
###########

## Specify additional locations of header files
## Your package locations should be listed before other locations
# include_directories(include)
include_directories(
  ${catkin_INCLUDE_DIRS}
)

## Declare a cpp library
# add_library(robot_spinner
#   src/${PROJECT_NAME}/robot_spinner.cpp
# )

## Declare a cpp executable
# add_executable(robot_spinner_node src/robot_spinner_node.cpp)

## Add cmake target dependencies of the executable/library
## as an example, message headers may need to be generated before nodes
# add_dependencies(robot_spinner_node robot_spinner_generate_messages_cpp)

## Specify libraries to link a library or executable target against
# target_link_libraries(robot_spinner_node
#   ${catkin_LIBRARIES}
# )

#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# install(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark executables and/or libraries for installation
# install(TARGETS robot_spinner robot_spinner_node
#   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
#   FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

#############
## Testing ##
#############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_robot_spinner.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
```

- **Complex and non-intuitive**
- **Template generated by `catkin_create_pkg` is very descriptive**
  - **Very long, includes most features of catkin**
  - **Typical CMakeLists.txt files only use a few features**
- **Typical workflows:**
  - **Uncomment and edit the relevant lines in the `catkin_create_pkg` template**
  - **Find a similar package and copy / edit their CMakeLists.txt file for your program**

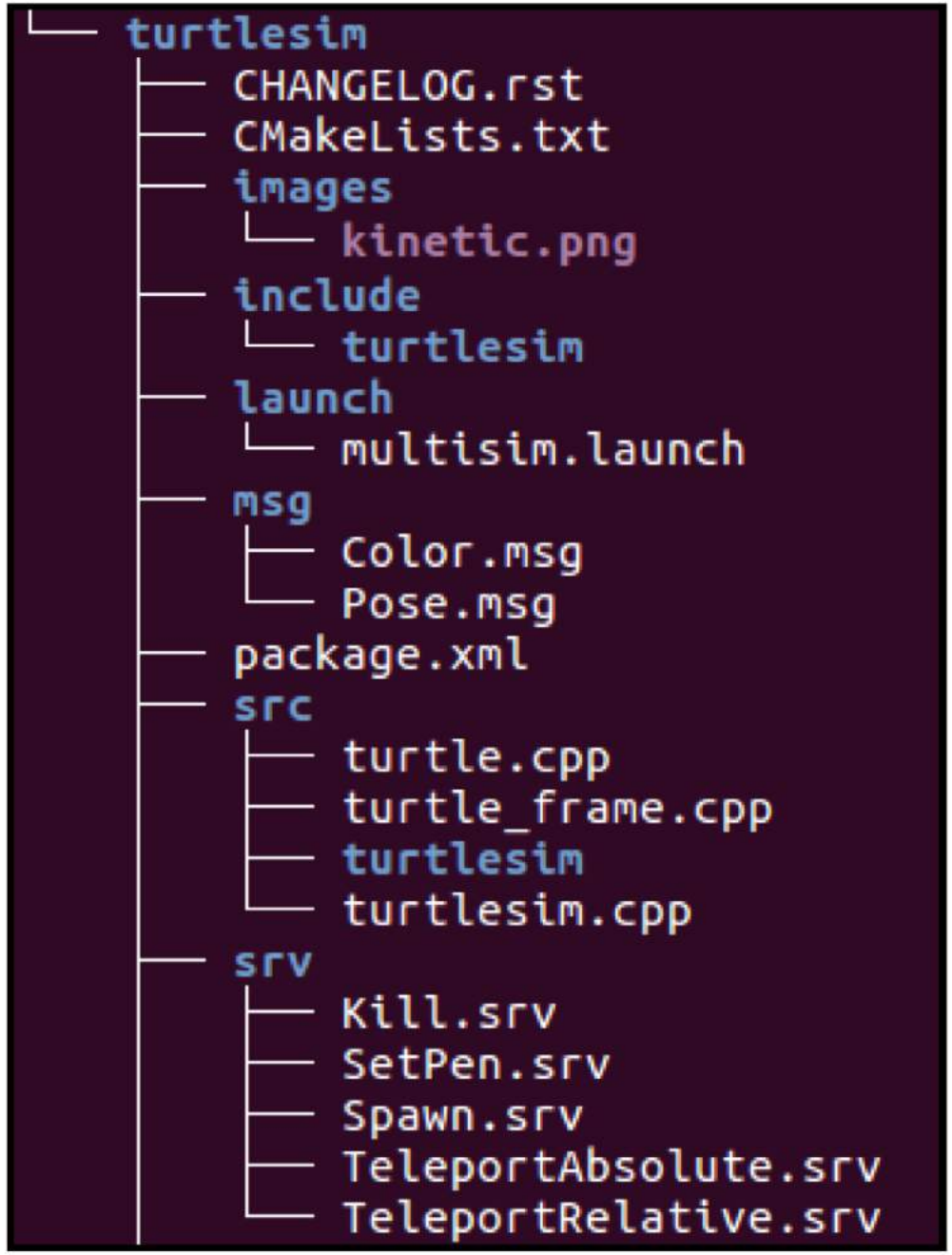
CMakeLists.txt Example (Python)

```
cmake_minimum_required(VERSION 2.8.3)
project(rospy_tutorials)

find_package(catkin REQUIRED COMPONENTS message_generation rostest
std_msgs)

add_message_files(DIRECTORY msg FILES Floats.msg HeaderString.msg)
add_service_files(DIRECTORY srv FILES AddTwoInts.srv BadTwoInts.srv)

generate_messages(DEPENDENCIES std_msgs)

catkin_package(CATKIN_DEPENDS message_runtime std_msgs)

add_rostest(test/test-add-two-ints.launch)

install(PROGRAMS
  005_add_two_ints/add_two_ints_client.py
  005_add_two_ints/add_two_ints_server.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION} )
```

CMakeLists.txt Example (C++)

```
cmake_minimum_required(VERSION 2.8.3)
project(roscpp_tutorials)

find_package(Boost REQUIRED COMPONENTS date_time thread)
find_package(catkin REQUIRED COMPONENTS message_generation rostime
roscpp rosconsole roscpp_serialization)

include_directories(${catkin_INCLUDE_DIRS})
link_directories(${catkin_LIBRARY_DIRS})

add_service_files(DIRECTORY srv FILES TwoInts.srv)
generate_messages(DEPENDENCIES std_msgs)

catkin_package(CATKIN_DEPENDS message_runtime std_msgs)

add_executable(babbler src/babbler.cpp)
target_link_libraries(babbler ${catkin_LIBRARIES} ${Boost_LIBRARIES})
add_dependencies(babbler roscpp_tutorials_gencpp)
install(TARGETS babbler
    RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```

Turtlesim Package



ROS code is grouped at two different levels:

- **Packages**
  A named collection of software that is built and treated as an atomic dependency in the ROS build system.
- **Stacks**
  A named collection of packages for distribution.

source code

header declarations

scripts

message definitions

service definitions

configuration files

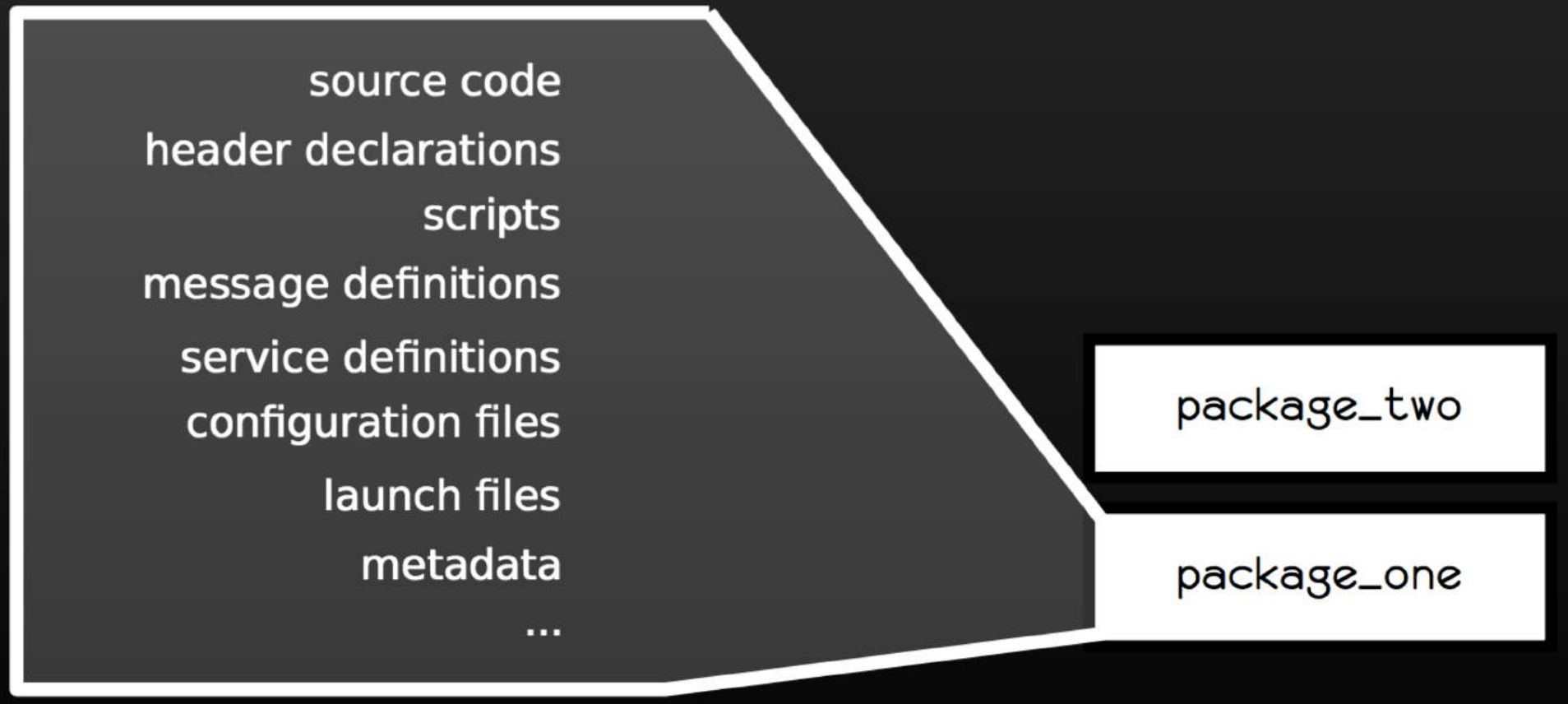
launch files

metadata

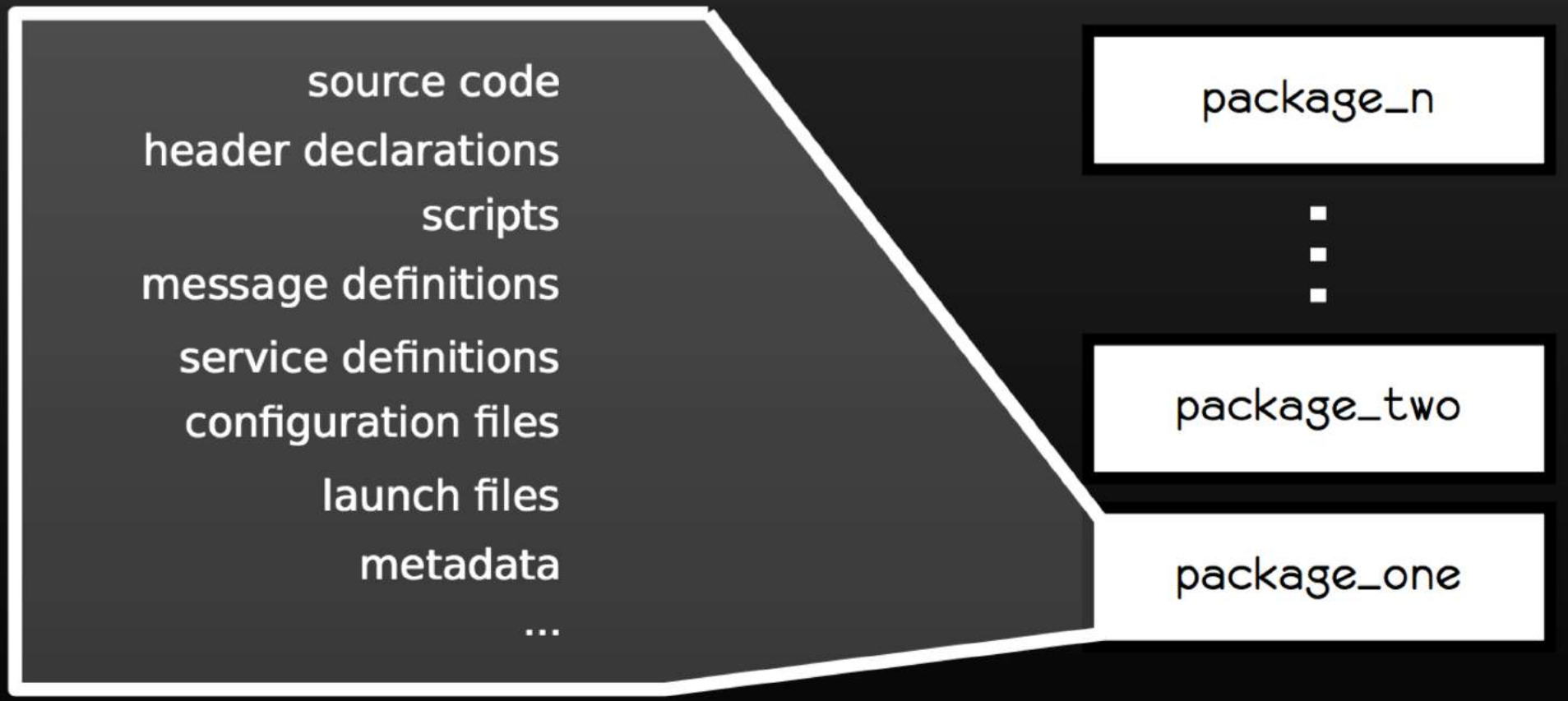
...


source code
header declarations
scripts
message definitions
service definitions
configuration files
launch files
metadata
...
package_one

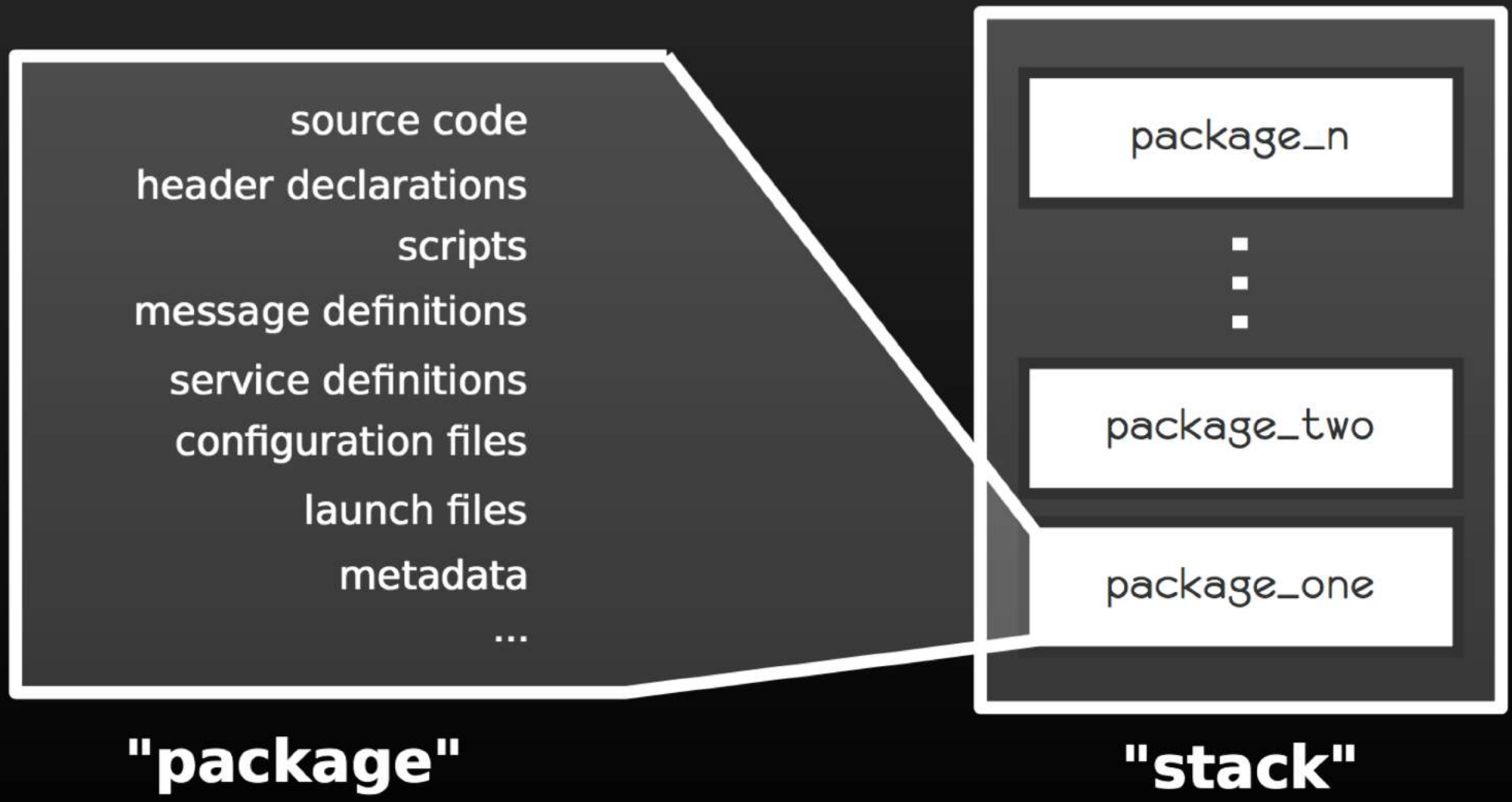
"package"


source code
header declarations
scripts
message definitions
service definitions
configuration files
launch files
metadata
...
package_two
package_one

"package"


source code
header declarations
scripts
message definitions
service definitions
configuration files
launch files
metadata
...
package_n
package_two
package_one

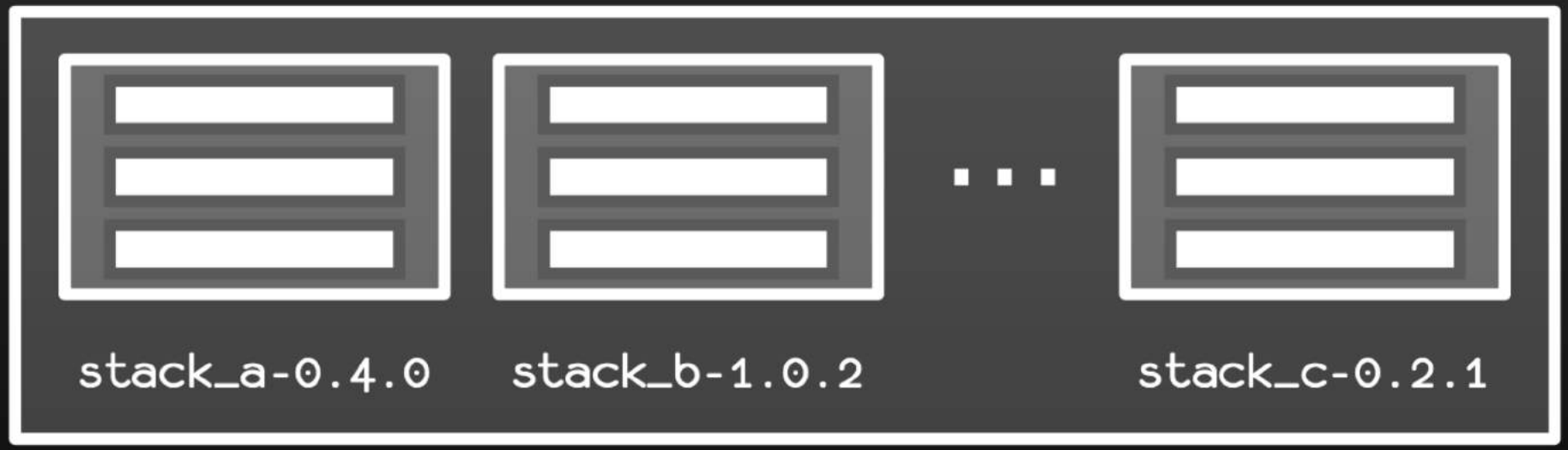
"package"


source code
header declarations
scripts
message definitions
service definitions
configuration files
launch files
metadata
...
"package"
package_n
package_two
package_one
"stack"

stack_a-0.4.0

stack_a-0.4.0 stack_b-1.0.2


stack_a-0.4.0
stack_b-1.0.2
stack_c-0.2.1



"distribution"