Summer School Robotics

Week 2 – Introduction to Ubuntu and Robot Operating System

Haoming Zhang, M.Sc.
Philipp Ennen, M.Sc.

IfU IMA ZLW RWTH AACHEN UNIVERSITY

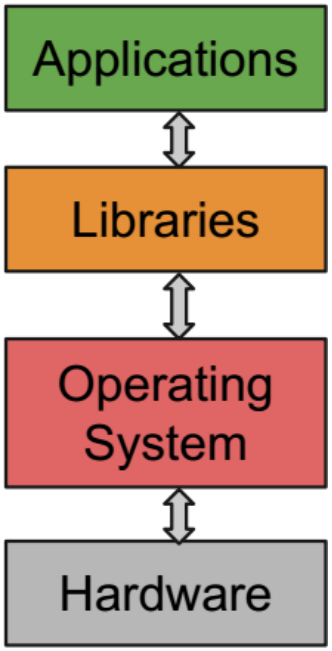
Outline

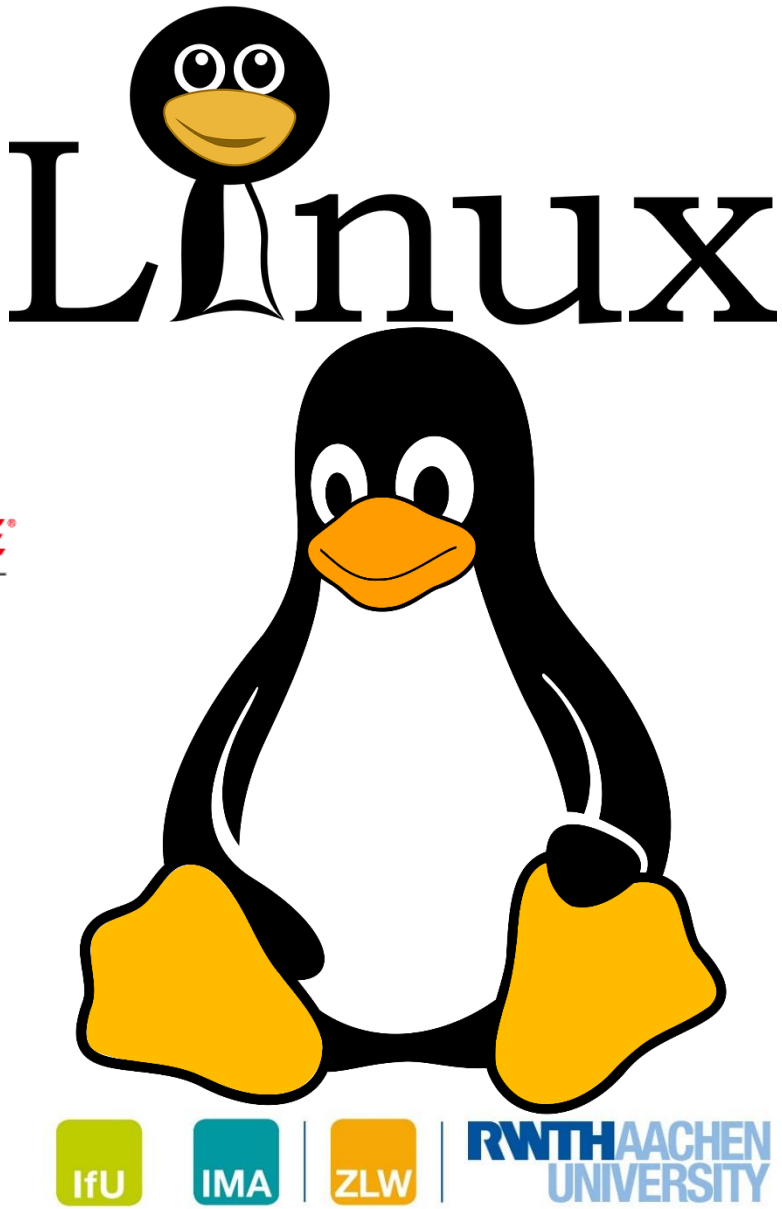
Outline

What is Linux?

A „UNIX-Like“ Operating System

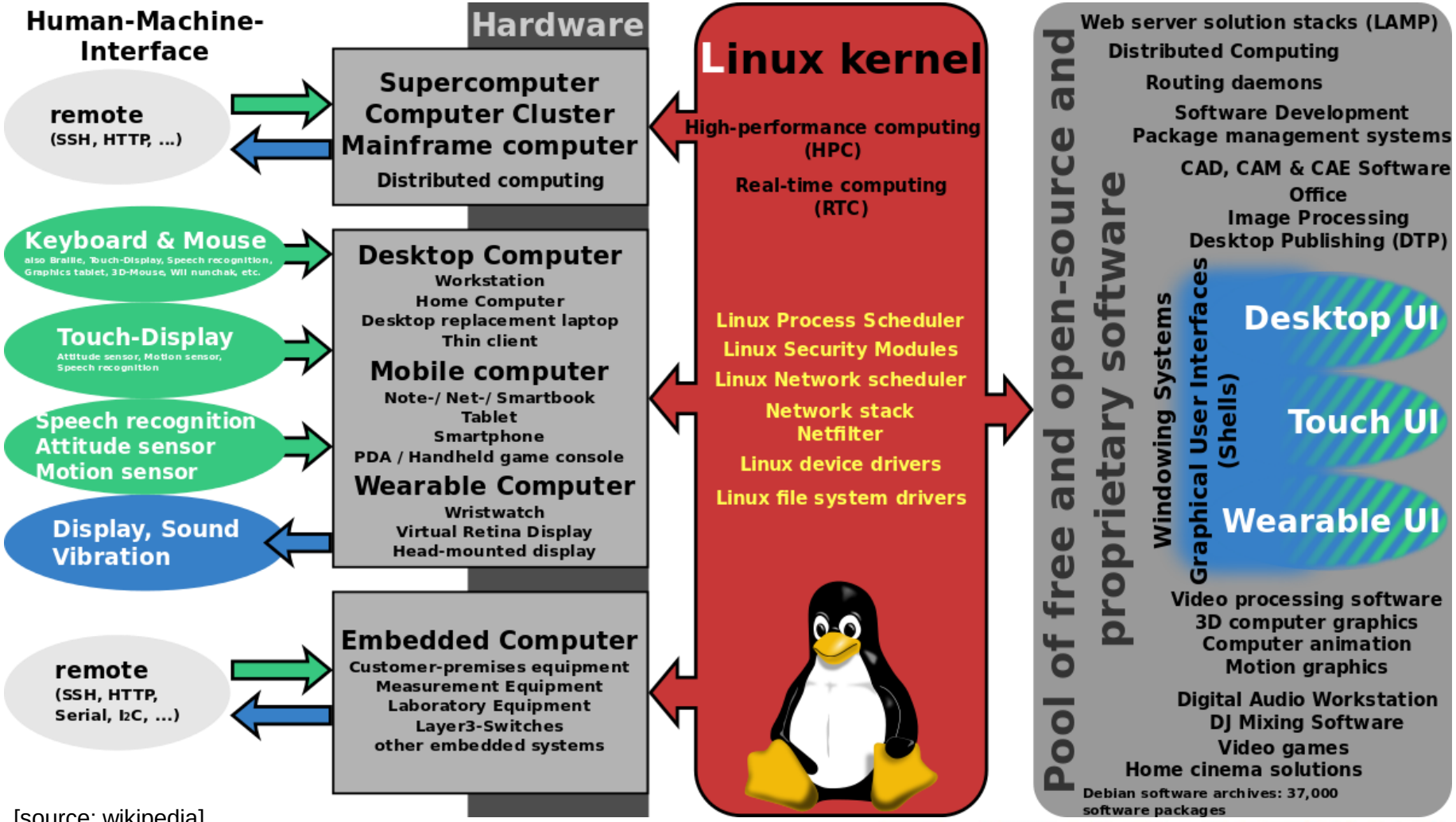
- Abstracts a set of hardware resources
  - High level interface instead of machine code
    e.g File storage from block devices
- Resource Management
  - Multiplexing (sharing) resources
    e.g Assign CPU time to applications





What is Linux?

Linux is made of:



[source: wikipedia]

Outline

Linux Histories and Distributions

Linux Histories

- Richard Stallman: Father of GNU Project in 1983

  creating a "complete Unix-compatible software system" composed entirely of free software

Richard Stallman

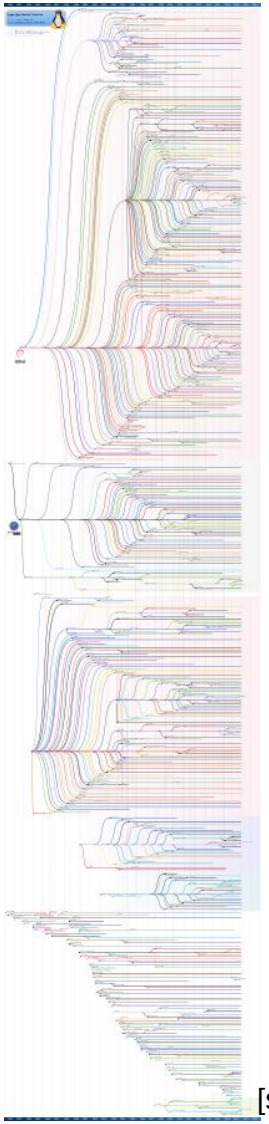
- Linux Torvalds: Father of Linux
  - Idea of a GNU Kernel, implemented in GNU C
  - Just for Fun: Story of an Accidental Revolutionary
  - 1st version supported Intel 80386
  - Currently various plattforms are supported

Linus Torvalds

Linux Histories and Distributions

Linux Distributions

[source: http://www.linux-netbook.com]



[source: wikipedia]

Outline

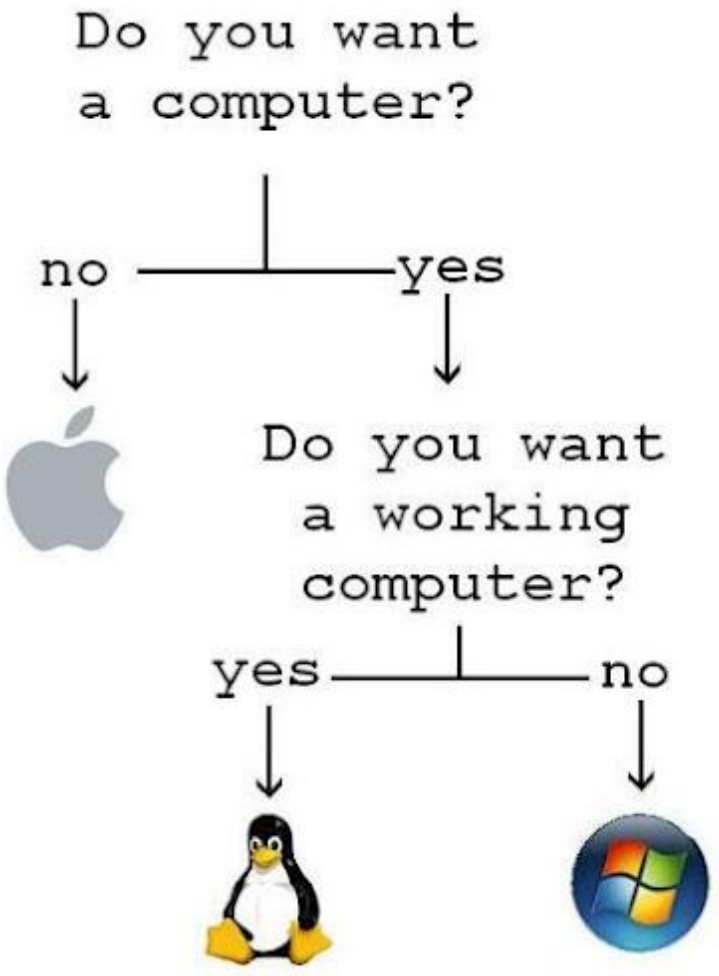
Linux: Pros and Cons

Pros:

- It‘s free.
- Stability.
- Easy to install.
- Work on any platforms.
- Less security risks.
- Many Flavors.
- Does not slow down over the time.
- Easy software installation.

Cons:

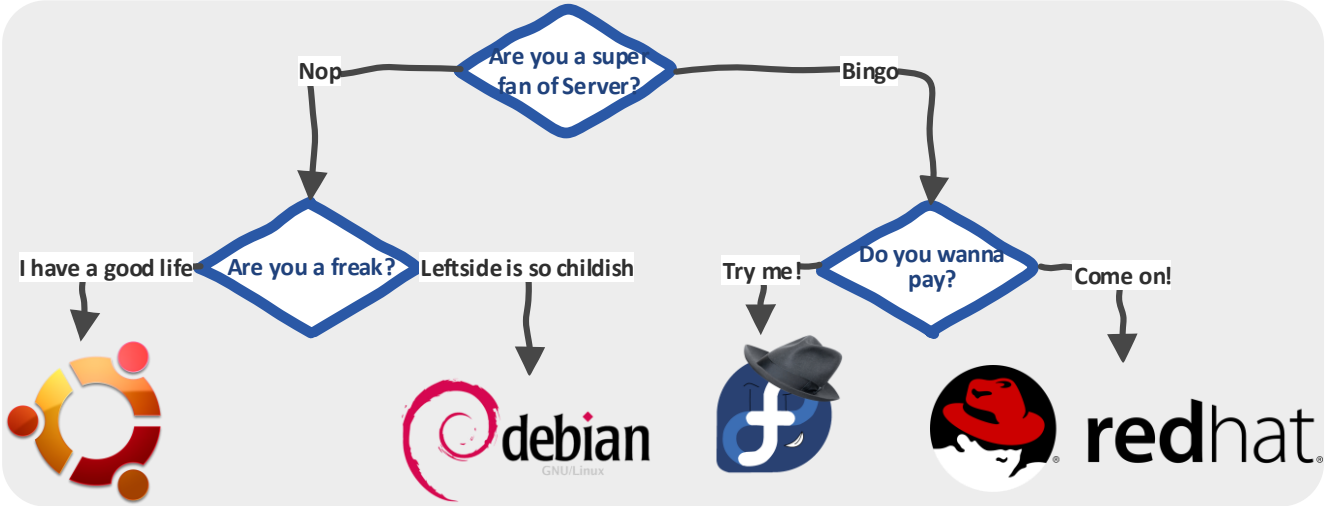
- Learning Curve.
- Non-Compatible Software.
- Technical support (unless you paid).



[source: http://iman.rastavibes.net/blog/index.php?article116/lulz-do-you-want-a-computer]

Why Ubuntu?

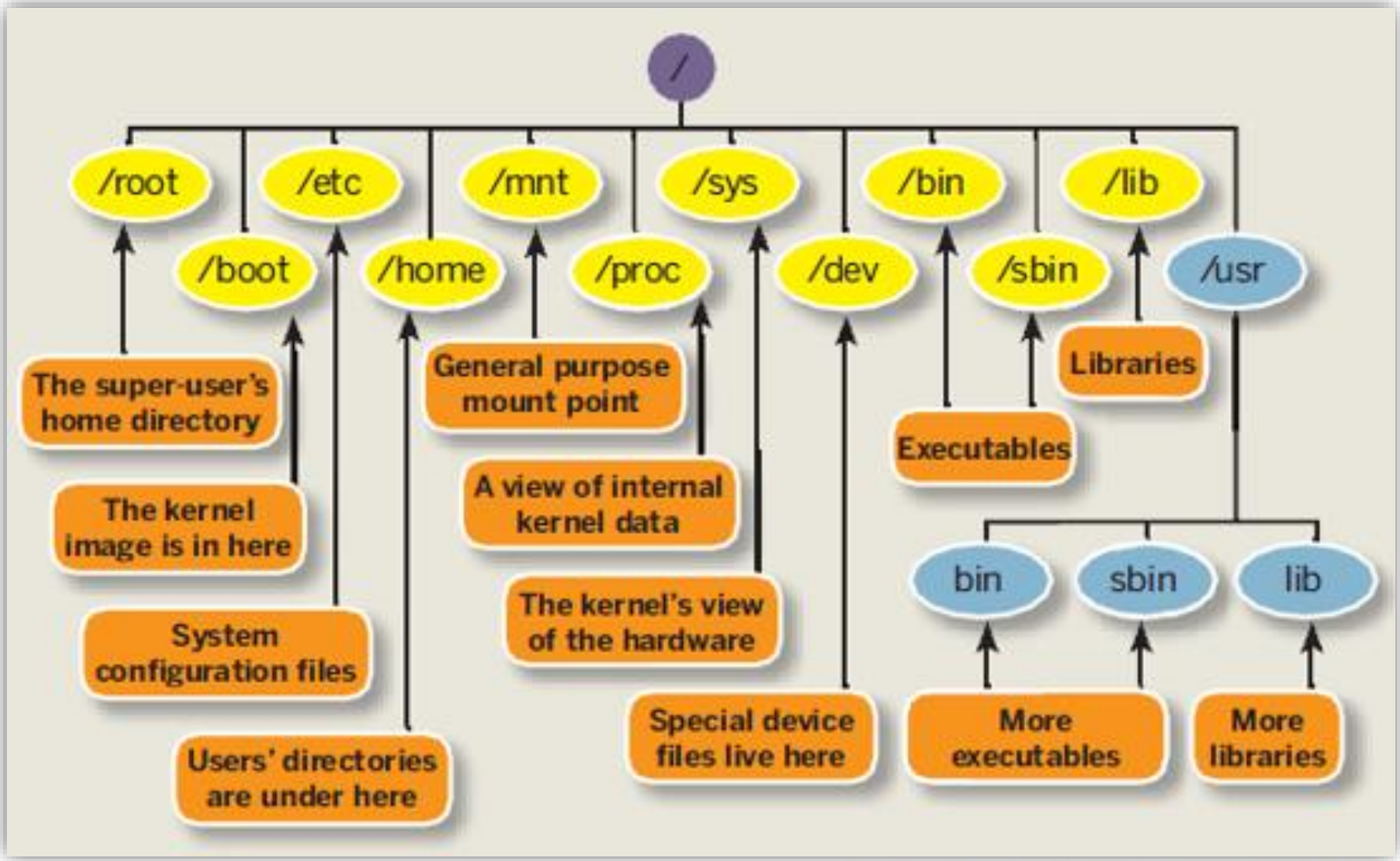
ubuntu



[source: http://funwithlinux.net]

Outline

Linux Filesystem Hierarchy



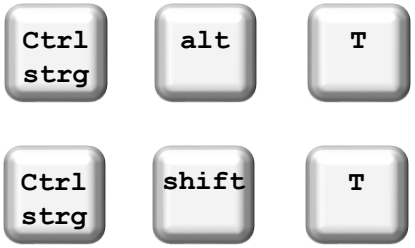
[source: https://askubuntu.com/tags/filesystem/info]

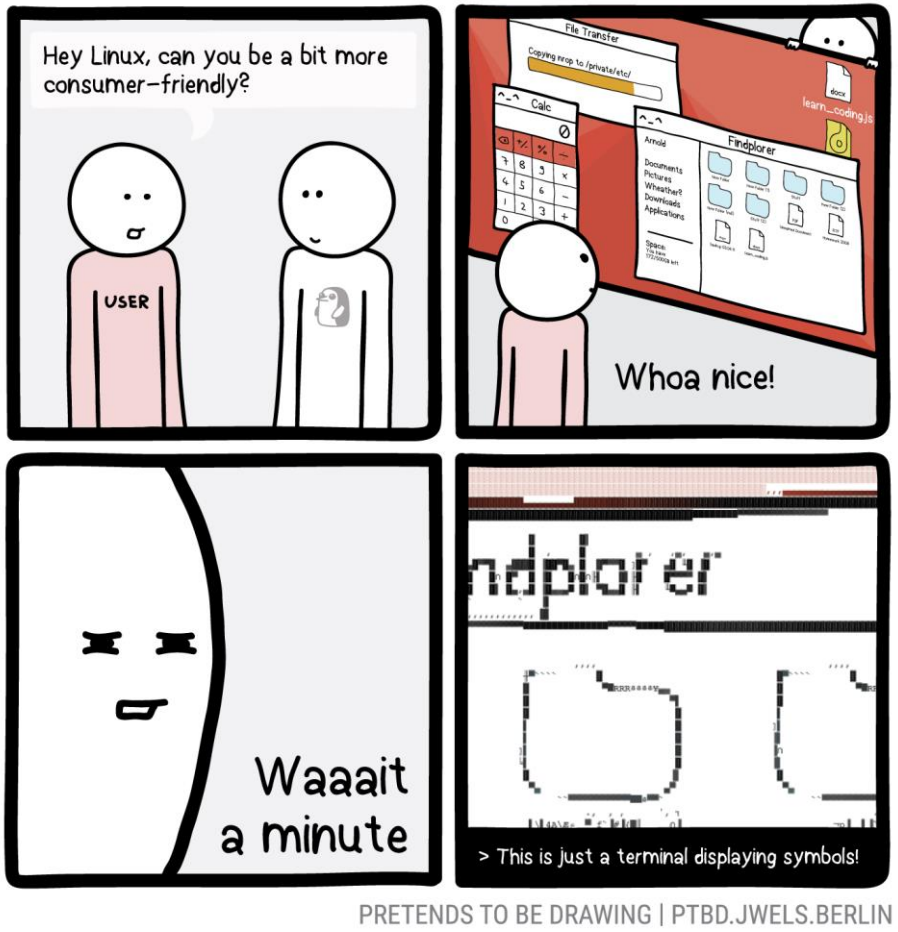
Outline

Linux Basics

- Your Working Partner:

  Terminal + Command Shell





Linux Basics

- Basical Commands
  - General

$ man [command] | man -k [expression]

- Display the manual pages
- man –k searches the expression in the Name of all commands

$ who | whoami

- Display all users
- Display the current user

$ sudo su

- Change to super user
- Using exit to escape

$ sudo reboot | sudo shutdown

- Reboot the OS
- Shutdown the OS

$ ps [-al] [-u user]

- Display running processes
- -a: all processes
- -l: long format display
- -u: all processes owned by a user

$ kill [-9] PID

- Terminates a process with PID
- -9: for „obstinate“ processes

Linux Basics

- Basical Commands
  - File Systems

$ pwd

- Display the current directory

$ cd [directory]

- Changes into the given directory
- ~: home
- /:root

$ ls [-alR] [file/dir.]

- List all files
- -a: all files also beginnt with .
- -R: recursively

$ mkdir [dir.] | rmdir [dir.]

- Creates directory
- Remove directiory

$ cp [-r] file1…n directory

- Copy without modify original
- -r: recursively

$ mv file/dir dir

- Rename / move file/dir to dir
- No [-r] required

Linux Basics

- Basical Commands
  - File Systems

```
$        rm [-irf] files/dirs
```

- Delete files / dirs
- -i: delete only after confirmation
- -f: force

```
$        cat file
```

- Display file on screen

```
$  chmod [ugoa] [+-=] [rwx] file/dir
```

- Changes permissions / access rights
- u: user, g: group, o:other users, a: all users
- r: read, w: modify, x: execute

For more: http://vic.gedris.org/Manual-ShellIntro/1.2/ShellIntro.pdf

Linux Basics

- Basical Commands
  - Something we need

```
$    sudo apt(-get) install [name]
```

- Install something which is in the current list

```
$              echo [variable]
```

- Display the variable
- Display a path: echo $

```
$  find dir. [-name,-iname,…] “name“
```

- Find something in dir with “name“

For more: http://scc.qibebt.cas.cn/docs/linux/script/LinuxCommand.pdf

Outline

Practicle Unit: Install a Text editor and create a workspace

- Install an awesome Text Editor: Sublime Text
  - Unfortunately not free
  - Visit: https://www.sublimetext.com/docs/3/linux_repositories.html

1. Install the GPG key:

```
$ wget -qO - https://download.sublimetext.com/sublimehq-pub.gpg | sudo apt-key add
```

2. Select a channel to use:

```
$ echo "deb https://download.sublimetext.com/ apt/stable/" |
  sudo tee /etc/apt/sources.list.d/sublime-text.list
```

3. Update Keys and install

```
$ sudo apt update |
  sudo apt install sublime-text
```

- Create Directories for later
  - Name: **`catkin_ws`** in **`/home`**
  - Name: **`src`** in **`/catkin_ws`**

Outline

Outline

Robot Operating System (ROS)

What is ROS?

Robot Software

The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.

Robot Operating System (ROS)

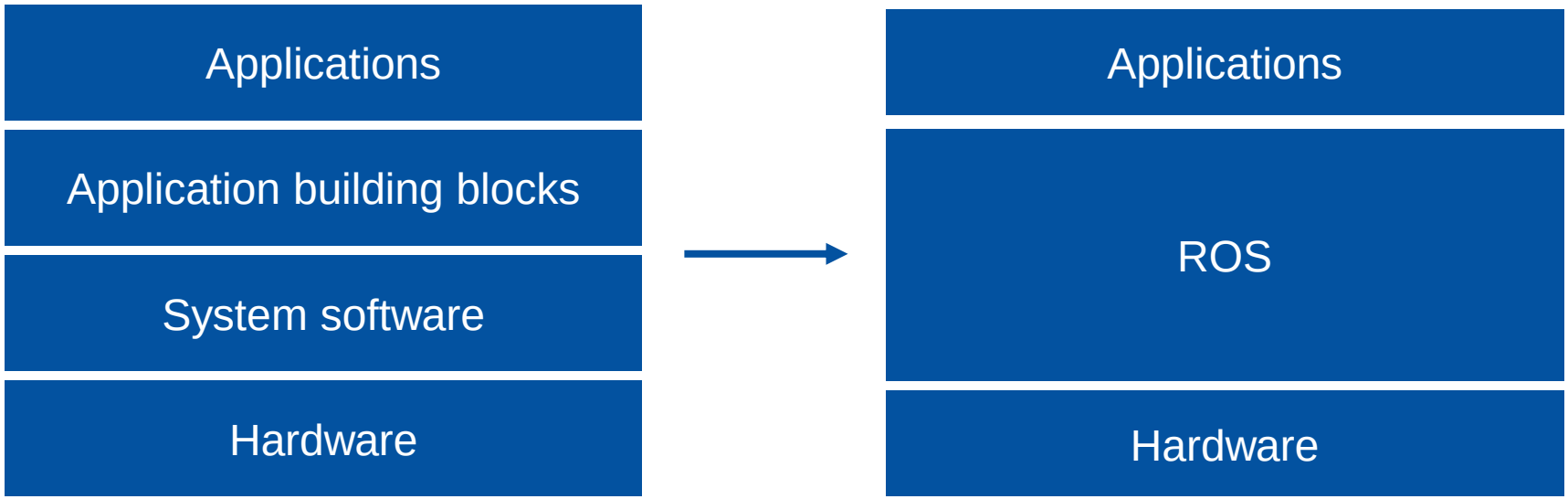
What is ROS?

- A “Meta” Operating System for robots
  - Open source
  - Runs in Linux (esp. Ubuntu)
  - Ongoing Windows implementation
- Agent based (nodes)
- Message passing
  - Publish
  - Subscribe
  - Services via remote invocation
- Supports numerous programming languages (C++. Python, Lisp, Java)

Robot Operating System (ROS)

What is ROS?

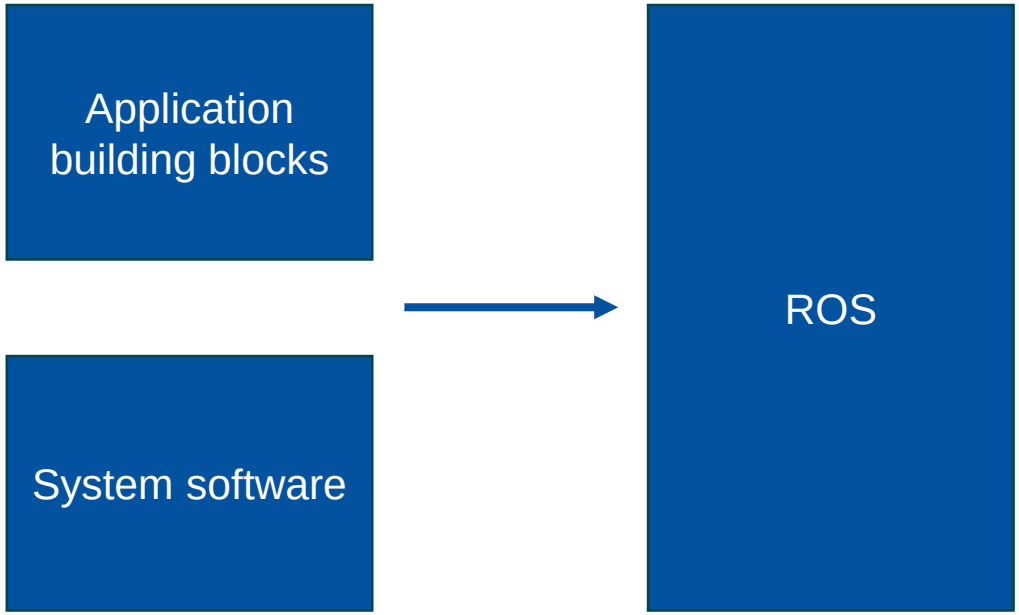
- Standardized layers
- System software abstracts
- Applications leverage other applications
- Widely existent sets of libraries



Robot Operating System (ROS)

What is ROS?

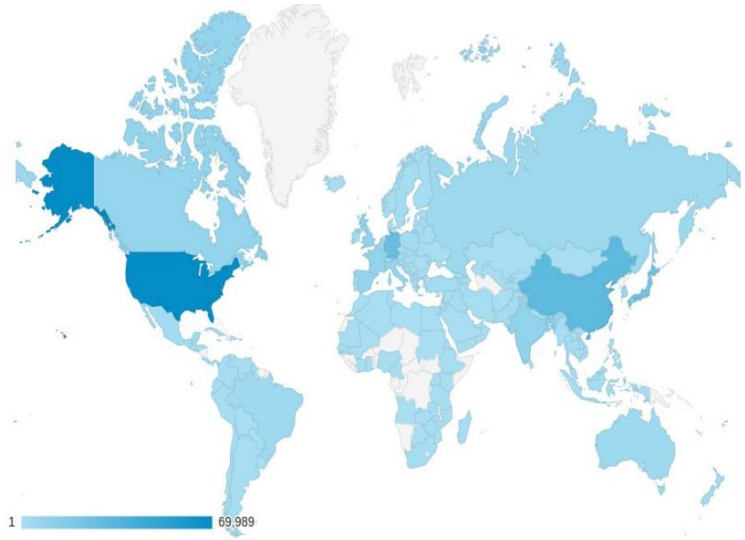
- Low level device abstraction
  - Joystick
  - GPS
  - Camera
  - Controllers
  - Laser Scanners
  - …
- Application building blocks
  - Coordinate system transforms
  - Visualization tools
  - Debugging tools
  - Robust navigation stack (SLAM with loop closure)
  - Arm path planning
  - Object recognition
  - …

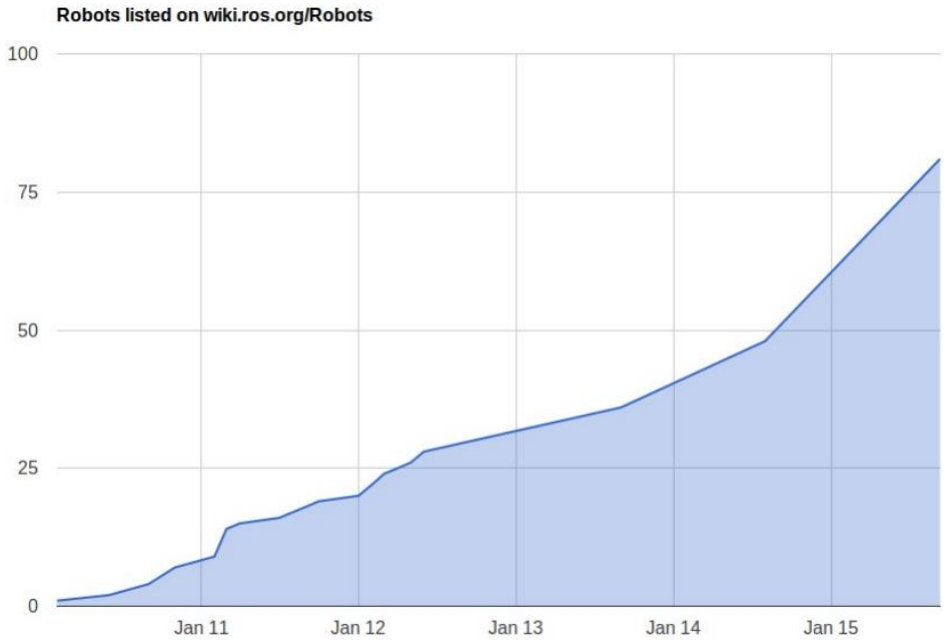


Robot Operating System (ROS)

Where is it used?

- More than 100 robots use ROS
  - http://wiki.ros.org/Robots
- Number of papers citing “ROS: an open-source Robot Operating System” (Quigley et al., 2009)
  - 3599





Outline

1	Introduction to Linux	14:00 – 14:45
2	Robot Operating System (ROS)	14:45 – 19:10
2.1	What is ROS?	14:45 – 14:55
2.2	ROS Basic	14:55 – 15:15
a	File System Concept	15:15 – 15:45
b	File System Practice	15:45 – 16:15
c	Computation Graph Level Concept	16:15 – 16:45
d	Computation Graph Level Practice	16:45 – 17:15
2.3	Get our Package and make it	17:15 – 17:45
2.4	3D Modeling and Simulation	17:45 – 18:00
2.5	Practical Unit: Create your first robot model	18:00 – 18:20
2.6	What is MoveIt!?	18:20 – 18:30
2.7	Practical Unit: Install MoveIt! and Build your Kinova in it	18:30 – 18:45
2.8	Practical Unit: Make the Kinova move	18:45 – 19:05
2.9	Conclusion and next Task	19:05 – 19:10

ROS Basics: Key Concepts

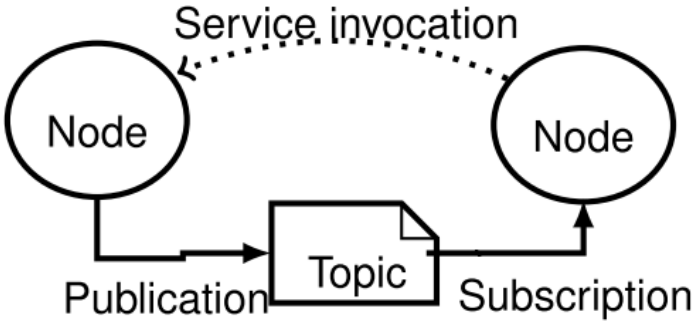
Key Concepts

- **roscore**:
  - **ROS Master**: Negotiates communication connections between nodes
  - **Parameter server**: stores persistent configuration parameters
  - **rosout**: a network-based stdout for human-readable messages

- **Package**: a virtual directory holding one or more executables (nodes)

- **Node**: An agent communicating with ROS and other nodes via
  - **Topics** (public/subscribe) using typed messages
  - **Services**: Request / Response paradigm (think of method or operation) via typed messages

- References:
  - ROS wiki: http://www.ros.org/wiki
  - ROS tutorials: http://www.ros.org/wiki/ROS/Tutorials/



Outline

1	Introduction to Linux		14:00 – 14:45
2	Robot Operating System (ROS)		14:45 – 19:10
	2.1	What is ROS?	14:45 – 14:55
	2.2	ROS Basic	14:55 – 15:15
		a File System Concept	15:15 – 15:45
		b File System Practice	15:45 – 16:15
		c Computation Graph Level Concept	16:15 – 16:45
		d Computation Graph Level Practice	16:45 – 17:15
	2.3	Get our Package and make it	17:15 – 17:45
	2.4	3D Modeling and Simulation	17:45 – 18:00
	2.5	Practical Unit: Create your first robot model	18:00 – 18:20
	2.6	What is MoveIt!?	18:20 – 18:30
	2.7	Practical Unit: Install MoveIt! and Build your Kinova in it	18:30 – 18:45
	2.8	Practical Unit: Make the Kinova move	18:45 – 19:05
	2.9	Conclusion and next Task	19:05 – 19:10

ROS Basics: File System

File System

- Packages
- Metapackages
- Package Manifest
- Repositories
- Message types
- Service types

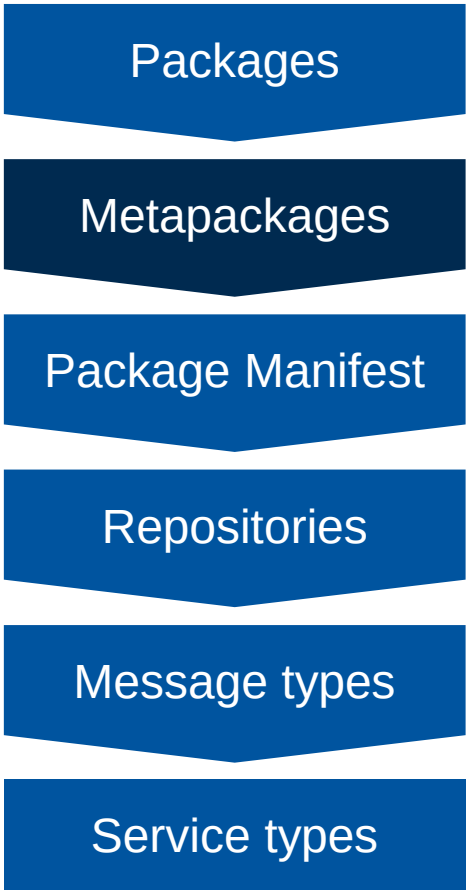
ROS Basics: File System

File System

- Main unit for organization
- A package may contain
  - Processes (nodes)
  - ROS-dependent libraries
  - Dataset
  - Configuration files (yaml, xml, …)
  - …
- Most atomic build item and release item

ROS Basics: File System

File System



- Specialized packages
- Represent a group of related other packages
- Used for backwards compatible placeholder
  - E.g. rosbuild stacks (http://wiki.ros.org/rosbuild)

ROS Basics: File System

File System

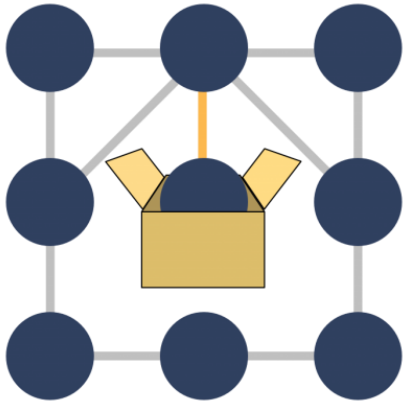
- package.xml
- Provide metadata about a package
  - name
  - version
  - description
  - license information
  - dependencies
  - …
- Defined in:
  - http://www.ros.org/reps/rep-0127.html

ROS Basics: File System

File System

- Packages
- Metapackages
- Package Manifest
- **Repositories**
- Message types
- Service types

- Collection of packages which share a common version control system (like git)
- These packages can be released together
  - catkin tool *bloom*
- Repositories can also contain only one package



ROS Basics: File System

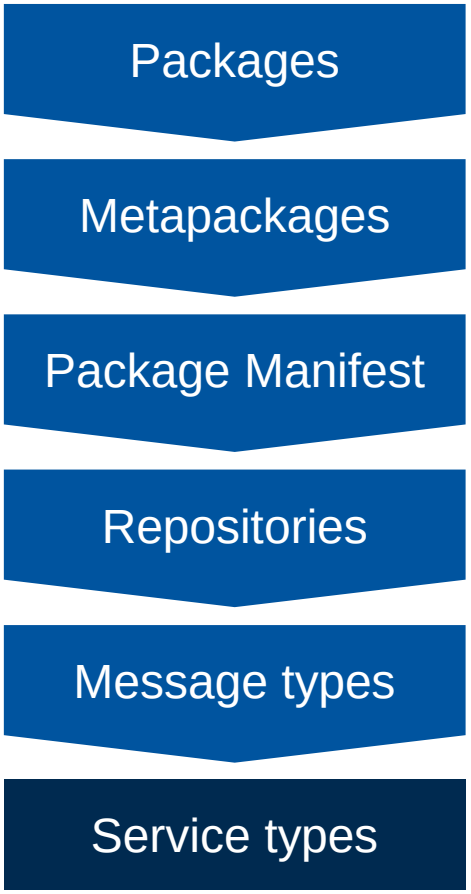
File System

- Message descriptions
- Stored in
  my_package/msg/MyMessageType.msg
- Data structure for messages sent in ROS

Primitive Type	Serialization	C++	Python
bool (1)	unsigned 8-bit int	uint8_t (2)	bool
int8	signed 8-bit int	int8_t	int
uint8	unsigned 8-bit int	uint8_t	int (3)
….	….	….	….
uint64	unsigned 64-bit int	uint64_t	long
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
string	ascii string (4)	std::string	str
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration

ROS Basics: File System

File System



- Service descriptions
- Stored in

  my_package/srv/MyServiceType.msg
- Request and response data structures

```
#request constants
int8 FOO=1
int8 BAR=2
#request fields
int8 foobar
another_pkg/AnotherMessage msg
---
#response constants
uint32 SECRET=123456
#response fields
another_pkg/YetAnotherMessage val
CustomMessageDefinedInThisPackage value
uint32 an_integer
```

request

response

You **cannot** embed another .srv inside of a .srv

Outline

1	Introduction to Linux		14:00 – 14:45
2	Robot Operating System (ROS)		14:45 – 19:10
2.1	What is ROS?		14:45 – 14:55
2.2	ROS Basic		14:55 – 15:15
	a	File System Concept	15:15 – 15:45
	b	File System Practice	15:45 – 16:15
	c	Computation Graph Level Concept	16:15 – 16:45
	d	Computation Graph Level Practice	16:45 – 17:15
2.3	Get our Package and make it		17:15 – 17:45
2.4	3D Modeling and Simulation		17:45 – 18:00
2.5	Practical Unit: Create your first robot model		18:00 – 18:20
2.6	What is MoveIt!?		18:20 – 18:30
2.7	Practical Unit: Install MoveIt! and Build your Kinova in it		18:30 – 18:45
2.8	Practical Unit: Make the Kinova move		18:45 – 19:05
2.9	Conclusion and next Task		19:05 – 19:10

ROS Basics: File System Practice

- Using *rospack*

```
$ rospack find [package_name]
```

- Find the following packages:
  - *roscpp*
  - *turtlesim*

absolute path to roscpp →
absolute path to turtlesim →

```
laptop@mobile-98:~$ rospack find roscpp
/opt/ros/kinetic/share/roscpp
laptop@mobile-98:~$ rospack find turtlesim
/opt/ros/kinetic/share/turtlesim
```

- Autocompletion makes things easier

```
$ rospack find turtle<tab><tab>
```

suggestions →

```
laptop@mobile-98:~$ rospack find turtle
turtle_actionlib  turtlesim        turtle_tf        turtle_tf2
laptop@mobile-98:~$ rospack find turtle
```

http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem

ROS Basics: File System Practice

- Using *rospack*

```
$ rospack depends [package_name]
```

- Show the dependencies of the following package:
  - *turtlesim*

for time and duration primitives →

including common message types (e.g. bool, byte, int, float, …) →

messages for common geometric primitives (e.g. Pose2D, Twist, Quaternion, …) →

```
laptop@mobile-98:~$ rospack depends turtlesim
cpp_common
rostime
roscpp_traits
roscpp_serialization
catkin
genmsg
genpy
message_runtime
std_msgs
geometry_msgs
gencpp
geneus
gennodejs
genlisp
message_generation
rosbuild
rosconsole
rosgraph_msgs
xmlrpcpp
roscpp
rospack
roslib
std_srvs
```

http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem

ROS Basics: File System Practice

- Using *roscd*

```
$ roscd [locationname[/subdir]]
```

- Change into the following directories with *roscd*:
  - *roscpp*
  - *turtlesim*

pwd: **p**rint **w**orking **d**irectory

current path →

```
laptop@mobile-98:~$ roscd turtlesim/
laptop@mobile-98:/opt/ros/kinetic/share/turtlesim$ pwd
/opt/ros/kinetic/share/turtlesim
```

- Autocompletion makes things easier

```
$ roscd turtle<tab><tab>
```

suggestions →

```
laptop@mobile-98:~$ roscd turtle
turtle_actionlib/  turtlesim/        turtle_tf/        turtle_tf2/
laptop@mobile-98:~$ roscd turtle
```

http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem

ROS Basics: File System Practice

- Note that *roscd*, like other ROS tools, will *only* find ROS packages that are within the directories listed in your *ROS_PACKAGE_PATH*.
- To see what is in your *ROS_PACKAGE_PATH*, type:

```
$ echo $ROS_PACKAGE_PATH
```

```
laptop@mobile-98:/opt/ros/kinetic/share/turtlesim$ echo $ROS_PACKAGE_PATH
/opt/ros/kinetic/share
```

- You can also enter subdirectories:

```
$ roscd turtlesim/srv/
```

current path →

```
laptop@mobile-98:/opt/ros/kinetic/share/turtlesim$ roscd turtlesim/srv/
laptop@mobile-98:/opt/ros/kinetic/share/turtlesim/srv$ pwd
/opt/ros/kinetic/share/turtlesim/srv
```

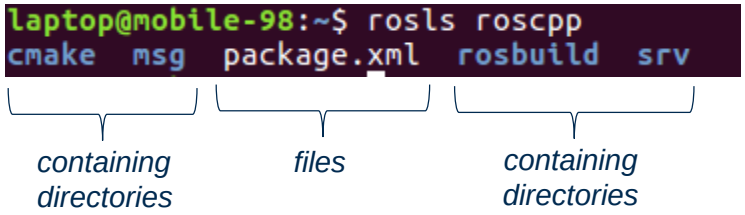
http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem

ROS Basics: File System Practice

- *rosls* is part of the rosbash suite
- It allows you to *ls* directly in a package by name rather than by absolute path
- Usage:

```
$ rosls [locationname[/subdir]]
```

- Test it for:
  - *roscpp*
  - *turtlesim*



http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem

Outline

1	Introduction to Linux	14:00 – 14:45
2	Robot Operating System (ROS)	14:45 – 19:10
2.1	What is ROS?	14:45 – 14:55
2.2	ROS Basic	14:55 – 15:15
a	File System Concept	15:15 – 15:45
b	File System Practice	15:45 – 16:15
c	Computation Graph Level Concept	16:15 – 16:45
d	Computation Graph Level Practice	16:45 – 17:15
2.3	Get our Package and make it	17:15 – 17:45
2.4	3D Modeling and Simulation	17:45 – 18:00
2.5	Practical Unit: Create your first robot model	18:00 – 18:20
2.6	What is MoveIt!?	18:20 – 18:30
2.7	Practical Unit: Install MoveIt! and Build your Kinova in it	18:30 – 18:45
2.8	Practical Unit: Make the Kinova move	18:45 – 19:05
2.9	Conclusion and next Task	19:05 – 19:10

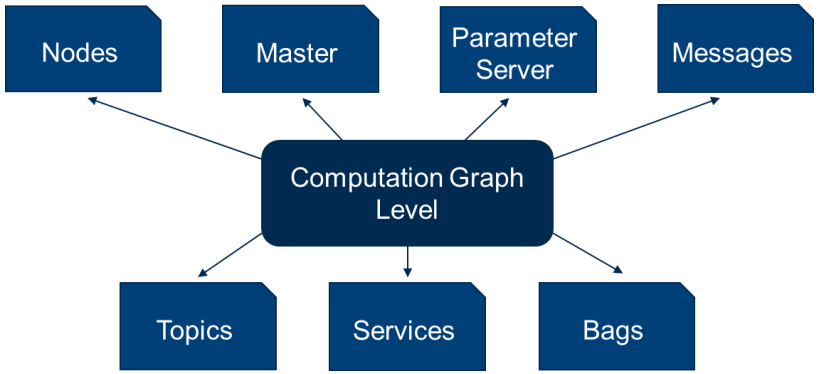
ROS Basics: Computation Graph Level

- The computation graph is the peer-to-peer network of ROS processes
- All provide data to the Graph in different ways
- Contained in the *ros_comm* repository
  - Several packages
  - ROS middleware/communications
- Core client libraries
  - roscpp, rospy, roslisp
- Graph introspection tools
  - rostopic, rosnode, rosservice, rosparam
  - e.g.

```
$ rostopic list
```

```
$ rostopic echo /cmd_vel
```

```
$ rosnode info /teleop_turtle
```

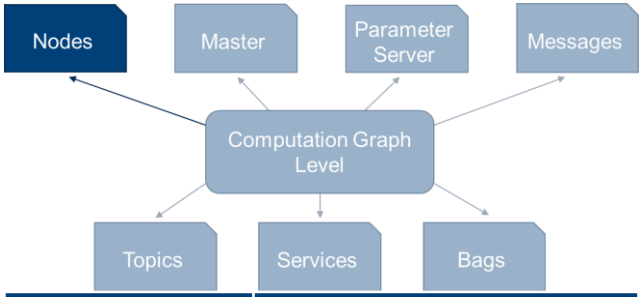


http://wiki.ros.org/ROS/Concepts : http://wiki.ros.org/ros_comm

ROS Basics: Computation Graph Level

- A *node* is a process that performs computation
- Combined together into a graph
- Communication
  - Topics
  - Services
  - Parameter Server
- E.g.:
  - One Node controls a laser range-finder
  - One Node controls the robot‘s wheel motors
  - One Node performs localization
  - …
- Fault tolerance
  - Crashes are isolated to individual nodes
- All running nodes have a *graph resource name*
  - Uniquely identification
  - E.g.: */hokuyo_node , /laser_node , /amcl*

http://wiki.ros.org/ROS/Concepts : http://wiki.ros.org/Nodes

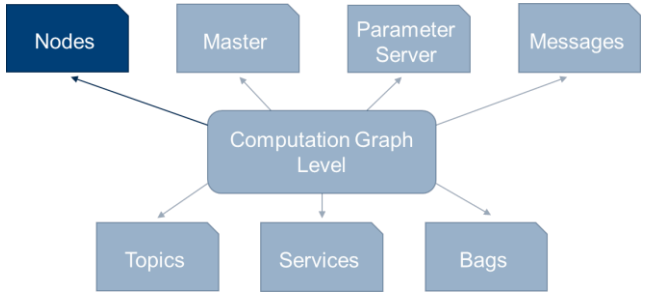


Commands	Description
$ rosnode ping	test connectivity to node
$ rosnode list	list active nodes
$ rosnode info	print information about node
$ rosnode machine	list node running on a particular machine or list machines
$ rosnode kill	kill a running node
$ rosnode cleanup	purge registration information of unreachable nodes

```
laptop@mobile-98:~$ rosnode info /teleop_turtle
--------------------------------------------------
Node [/teleop_turtle]
Publications:
 * /rosout [rosgraph_msgs/Log]
 * /turtle1/cmd_vel [geometry_msgs/Twist]

Subscriptions: None

Services:
 * /teleop_turtle/get_loggers
 * /teleop_turtle/set_logger_level
```

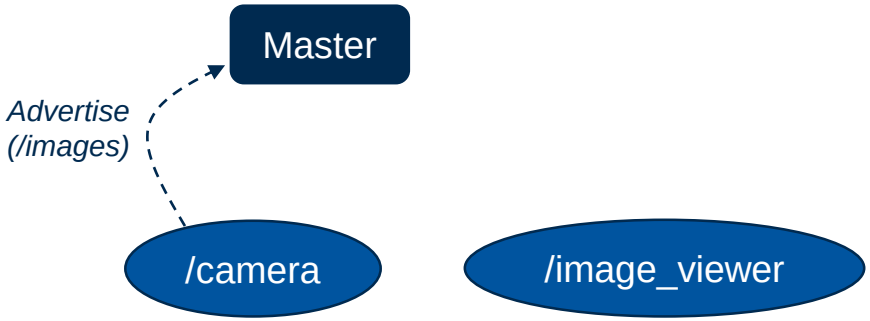
ROS Basics: Computation Graph Level

- The *ROS master* provides naming and registration services to the rest of the *nodes*
  - Tracks *publishers, subscribers, services*
- Enable individual ROS *nodes* to locate each other
  - Subsequent, they communicate peer-to-peer
- Provides the *Parameter Server*
- *Example: Two nodes: /camera and /image_viewer*



$ **roscore**

```
laptop@mobile-98:~$ roscore
... logging to /home/laptop/.ros/log/b3a4f352-1a27-11e7-8a27-
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://mobile-98:44845/
ros_comm version 1.12.7


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.7

NODES

auto-starting new master
process[master]: started with pid [8919]
ROS_MASTER_URI=http://mobile-98:11311/

setting /run_id to b3a4f352-1a27-11e7-8a27-6067201943ac
process[rosout-1]: started with pid [8932]
started core service [/rosout]
```

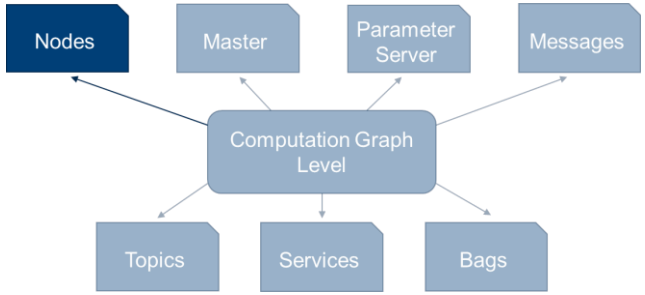


http://wiki.ros.org/ROS/Concepts : http://wiki.ros.org/Master

IfU IMA ZLW RWTH AACHEN UNIVERSITY

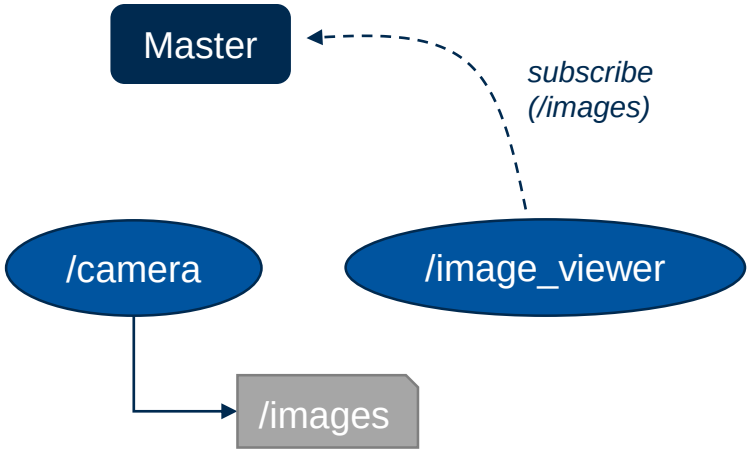
ROS Basics: Computation Graph Level

- The *ROS master* provides naming and registration services to the rest of the *nodes*
  - Tracks *publishers, subscribers, services*
- Enable individual ROS *nodes* to locate each other
  - Subsequent, they communicate peer-to-peer
- Provides the *Parameter Server*
- *Example: Two nodes: /camera and /image_viewer*




```
$ roscore
```

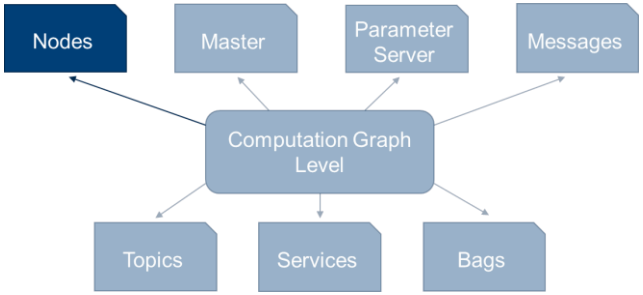
```
laptop@mobile-98:~$ roscore
... logging to /home/laptop/.ros/log/b3a4f352-1a27-11e7-8a27-
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://mobile-98:44845/
ros_comm version 1.12.7


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.7

NODES

auto-starting new master
process[master]: started with pid [8919]
ROS_MASTER_URI=http://mobile-98:11311/

setting /run_id to b3a4f352-1a27-11e7-8a27-6067201943ac
process[rosout-1]: started with pid [8932]
started core service [/rosout]
```



http://wiki.ros.org/ROS/Concepts : http://wiki.ros.org/Master

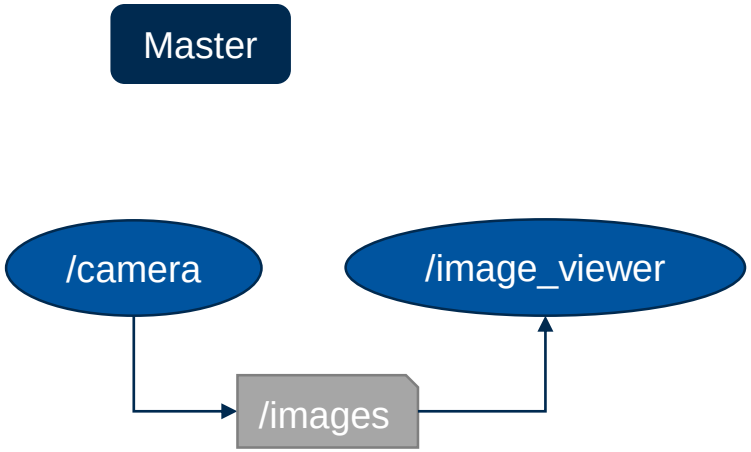
ROS Basics: Computation Graph Level

- The *ROS master* provides naming and registration services to the rest of the *nodes*
  - Tracks *publishers, subscribers, services*
- Enable individual ROS *nodes* to locate each other
  - Subsequent, they communicate peer-to-peer
- Provides the *Parameter Server*
- *Example: Two nodes: /camera and /image_viewer*




```
$ roscore
```

```
laptop@mobile-98:~$ roscore
... logging to /home/laptop/.ros/log/b3a4f352-1a27-11e7-8a27-
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://mobile-98:44845/
ros_comm version 1.12.7


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.7

NODES

auto-starting new master
process[master]: started with pid [8919]
ROS_MASTER_URI=http://mobile-98:11311/

setting /run_id to b3a4f352-1a27-11e7-8a27-6067201943ac
process[rosout-1]: started with pid [8932]
started core service [/rosout]
```

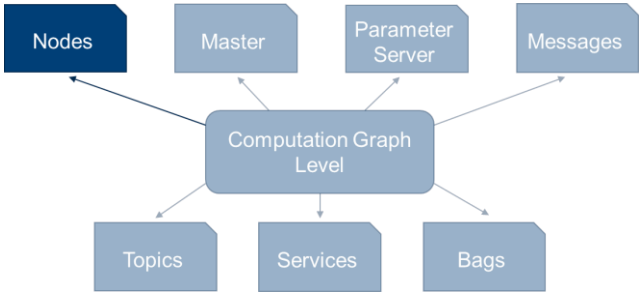


http://wiki.ros.org/ROS/Concepts : http://wiki.ros.org/Master

IfU IMA ZLW RWTH AACHEN UNIVERSITY

ROS Basics: Computation Graph Level

- A *shared* multivariate dictionary
- *Usage:*
  - *Store and retrieve parameters at runtime (nodes)*
  - *Static, non-binary data (configuration parameters)*
- Meant to be globally viewable (*rosparam*)
- Implemented using *XMLRPC*
  - *Remote procedure call*
  - *Encode via XML*
  - *Uses HTTP as a transport mechanism*
  - *Data types: booleans, strings, doubles, lists, …*
  - *Dictionaries (structs) are also possible!*
- Normal ROS naming convention



Commands	Description
$ rosparam set	set parameter
$ rosparam get	get parameter
$ rosparam load	load parameters from file
$ rosparam dump	dump parameters to file
$ rosparam delete	delete parameter
$ rosparam list	list parameter names

```
laptop@mobile-98:~$ rosparam list
/background_b
/background_g
/background_r
/rosdistro
/roslaunch/uris/host_mobile_98__44845
/rosversion
/run_id
laptop@mobile-98:~$ rosparam get /background_b
255
laptop@mobile-98:~$ rosparam get /background_g
86
laptop@mobile-98:~$ rosparam get /background_r
69
```

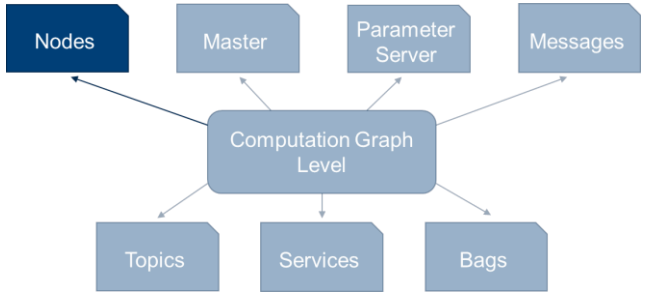
http://wiki.ros.org/ROS/Concepts : http://wiki.ros.org/Parameter%20Server

ROS Basics: Computation Graph Level

- *Nodes* communicating with each other by publishing **messages** to topics
- A **message** is a simple data structure
  - Comprising typed fields
  - Primitive types: integer, floating point, boolean, …
  - Arrays of primitive types
  - Arbitrarily nested structures and arrays
- *Nodes* can also exchange a request and response
  - Part of *ROS service call*
  - Defined in *srv files*
- Naming convention
  - [package name] + / + [name of .msg file]
  - E.g. *std_msgs/msg/String.msg*
- *MD5 checksum*
  - Versionization by MD5 sum of .msg file
  - Message type and MD5 sum have to match

```
5f3f794301c7af61b3beab5b9997bb64  PoseArray.msg
```

md5sum



Commands	Description
$ rosmsg show	show message description
$ rosmsg info	alias for rosmsg show
$ rosmsg list	list all messages
$ rosmsg md5	display message md5sum
$ rosmsg package	list messages in a package
$ rosmsg packages	list packages that contain messages

```
laptop@mobile-98:~$ rosmsg list
actionlib/TestAction
actionlib/TestActionFeedback
actionlib/TestActionGoal
actionlib/TestActionResult
actionlib/TestFeedback
geometry_msgs/Pose2D
geometry_msgs/PoseArray
geometry_msgs/PoseStamped
geometry_msgs/PoseWithCovariance
geometry_msgs/PoseWithCovarianceStamped
geometry_msgs/Quaternion
```

http://wiki.ros.org/ROS/Concepts : http://wiki.ros.org/Messages

ROS Basics: Computation Graph Level

- *Example message composition (geometry_msgs):*

Type: ***PoseArray.msg*** *contains Array of Pose*

```
laptop@mobile-98:/opt/ros/kinetic/share/geometry_msgs/msg$ cat PoseArray.msg
# An array of poses with a header for global reference.

Header header

Pose[] poses
```

Type: ***Pose.msg*** *contains Point & Quaternion*

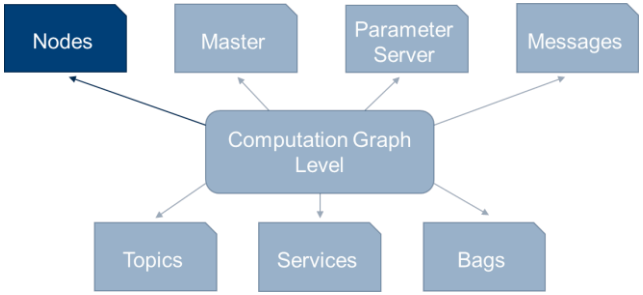
```
laptop@mobile-98:/opt/ros/kinetic/share/geometry_msgs/msg$ cat Pose.msg
# A representation of pose in free space, composed of position and orientation.
Point position
Quaternion orientation
```

Type: ***Point.msg*** *contains 3x float64*

```
laptop@mobile-98:/opt/ros/kinetic/share/geometry_msgs/msg$ cat Point.msg
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

Type: ***Quaternion.msg*** *contains 4x float64*

```
laptop@mobile-98:/opt/ros/kinetic/share/geometry_msgs/msg$ cat Quaternion.msg
# This represents an orientation in free space in quaternion form.

float64 x
float64 y
float64 z
float64 w
```



Commands	Description
$ rosmsg show	show message description
$ rosmsg info	alias for rosmsg show
$ rosmsg list	list all messages
$ rosmsg md5	display message md5sum
$ rosmsg package	list messages in a package
$ rosmsg packages	list packages that contain messages

```
laptop@mobile-98:~$ rosmsg list
actionlib/TestAction
actionlib/TestActionFeedback
actionlib/TestActionGoal
actionlib/TestActionResult
actionlib/TestFeedback
geometry_msgs/Pose2D
geometry_msgs/PoseArray
geometry_msgs/PoseStamped
geometry_msgs/PoseWithCovariance
geometry_msgs/PoseWithCovarianceStamped
geometry_msgs/Quaternion
geometry_msgs/QuaternionStamped
```

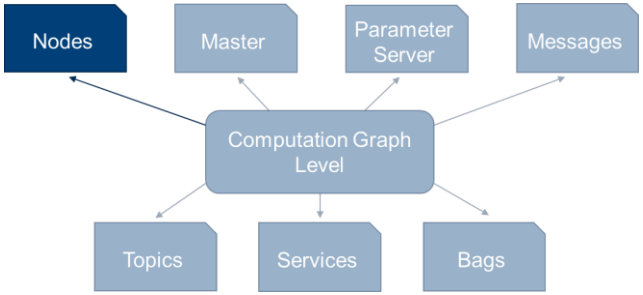
http://wiki.ros.org/ROS/Concepts : http://wiki.ros.org/Messages

IfU IMA ZLW RWTH AACHEN UNIVERSITY

ROS Basics: Computation Graph Level

- *Messages* are routed via transport system
  - *Publish / subscribe semantics*
  - *Decouples production of information from its consumption*
  - *Nodes are not awar of who they are communicating with*
- Publishing/Subscribing to *topics*
  - Nodes that are interested in data *subscribe*
  - Nodes that generate data *publish*
- Topic are intended for unidirectional, streaming communication
  - If you need request/response use ***services***
- Topic Types
  - *Each topic is strongly types by the ROS message*
  - *Master **does not** enforce type consistency*
- *Topic Transports*
  - ***TCPROS** (default): streams message data over persistent TCP/IP connections*
  - ***UDPROS**: separates messages into UDP packets*

http://wiki.ros.org/ROS/Concepts : http://wiki.ros.org/Topics

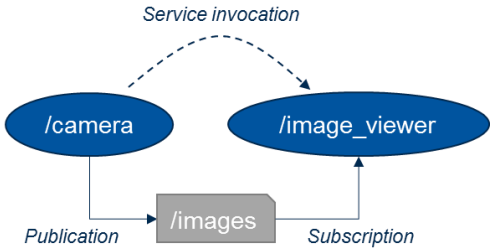


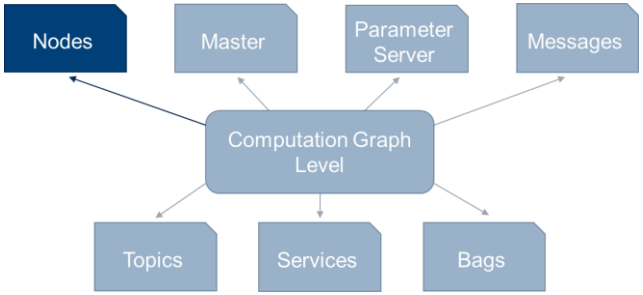
Commands	Description
$ rostopic bw	display bandwidth used by topic
$ rostopic delay	display delay of topic from timestamp in header
$ rostopic echo	print messages to screen
$ rostopic find	find topics by type
$ rostopic hz	display publishing rate of topic
$ rostopic info	print information about active topic
$ rostopic list	list active topics
$ rostopic pub	publish data to topic
$ rostopic type	print topic or field type

```
Type: turtlesim/Pose

Publishers:
 * /turtlesim (http://mobile-98:39731/)

Subscribers:
 * /rostopic_9360_1491414592736 (http://mobile-98:33807/)
```

ROS Basics: Computation Graph Level

- Publish/Subscribe is not appropriate for *RPC request/reply* interactions
  - Often required in distributed systems
- Request/reply
  - Is done via a service
  - Defined by **a pair** of messages
    - one for the request
    - one for the reply
- Services
  - Defined using **srv** files
  - Compiled into source code by a ROS client library
- Services have an associated service type
  - like topics
  - package resource name of the .srv file
- Versioned by a MD5 sum of the .srv file





Commands	Description
$ rosservice args	print service arguments
$ rosservice call	call the service with provided args
$ rosservice find	find services by service type
$ rosservice info	print information about service
$ rosservice list	list active services
$ rosservice type	print service type
$ rosservice uri	print service ROSRPC uri

```
laptop@mobile-98:~$ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/rostopic_9360_1491414592736/get_loggers
/rostopic_9360_1491414592736/set_logger_level
/spawn
/teleop_turtle/get_loggers
```

http://wiki.ros.org/ROS/Concepts : http://wiki.ros.org/Services

ROS Basics: Computation Graph Level

- Some examples of .srv files:

  Get a map by request:

```
laptop@mobile-98:/opt/ros/kinetic/share$ cat ./map_msgs/srv/GetPointMap.srv
# Get the map as a sensor_msgs/PointCloud2
---
sensor_msgs/PointCloud2 map
```

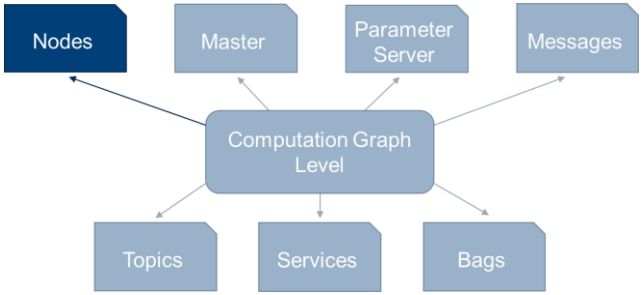
  Bilateral: setting and getting link states

  - get link state:

```
laptop@mobile-98:/opt/ros/kinetic/share$ cat ./gazebo_msgs/srv/GetLinkState.srv
string link_name          # name of link
                          # link names are prefixed by model name, e.g. pr2::base
string reference_frame    # reference frame of returned information, must be a va
                          # if empty, use inertial (gazebo world) frame
                          # reference_frame names are prefixed by model name, e.g
---
gazebo_msgs/LinkState link_state
bool success              # return true if get info is successful
string status_message     # comments if available
```

  - and set link state:

```
laptop@mobile-98:/opt/ros/kinetic/share$ cat ./gazebo_msgs/srv/SetLinkState.srv
gazebo_msgs/LinkState link_state
---
bool success                # return true if set wrench successful
string status_message       # comments if available
```



Commands	Description
$ rosservice args	print service arguments
$ rosservice call	call the service with provided args
$ rosservice find	find services by service type
$ rosservice info	print information about service
$ rosservice list	list active services
$ rosservice type	print service type
$ rosservice uri	print service ROSRPC uri

```
laptop@mobile-98:~$ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/rostopic_9360_1491414592736/get_loggers
/rostopic_9360_1491414592736/set_logger_level
/spawn
/teleop_turtle/get_loggers
```

http://wiki.ros.org/ROS/Concepts : http://wiki.ros.org/Services

ROS Basics: Computation Graph Level - Overview

- Quick overview of Graph Concepts

Concept	Description
Nodes	A node is an executable that uses ROS to communicate with other nodes.
Messages	ROS data type used when subscribing or publishing to a topic.
Topics	Nodes can *publish* messages to a topic as well as *subscribe* to a topic to receive messages.
Master	Name service for ROS (i.e. helps nodes find each other)
rosout	ROS equivalent of stdout/stderr
roscore	Master + rosout + parameter server (parameter server will be introduced later)

http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

Outline

ROS Basics: Computation Graph Level - Practice

- roscore is the first thing you should run when using ROS.

```
$ roscore
```

- Using rosnode:
  - *Open up a **new tab*** 

  - *rosnode displays information about the ROS nodes that are currently running. The rosnode list command lists these active nodes:*

```
$ rosnode list
```

  - *you will see:*

http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

ROS Basics: Computation Graph Level - Practice

- roscore is the first thing you should run when using ROS.

```
$ roscore
```

- Using rosnode:
  - *Open up a **new tab*** 

  - *rosnode displays information about the ROS nodes that are currently running. The rosnode list command lists these active nodes:*

```
$ rosnode list
```

  - *you will see:*

```
laptop@mobile-98:~$ rosnode list
/rosout
```

  - *This showed us that there is only one node running: rosout. This is always running as it collects and logs nodes' debugging output.*

http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

ROS Basics: Computation Graph Level - Practice

- The rosnode info command returns information about a specific node

```
$ rosnode info /rosout
```

- This gave us some more information about rosout, such as the fact that it publishes */rosout_agg*:

http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

ROS Basics: Computation Graph Level - Practice

- The rosnode info command returns information about a specific node

```
$ rosnode info /rosout
```

- This gave us some more information about rosout, such as the fact that it publishes */rosout_agg*:

```
laptop@mobile-98:~$ rosnode info /rosout
--------------------------------------------------------------------------------
Node [/rosout]
Publications:
 * /rosout_agg [rosgraph_msgs/Log]

Subscriptions:
 * /rosout [unknown type]

Services:
 * /rosout/set_logger_level
 * /rosout/get_loggers


contacting node http://mobile-98:42525/ ...
Pid: 16246
```

- Now, let's see some more nodes. For this, we're going to use rosrun to bring up another node

http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

ROS Basics: Computation Graph Level - Practice

- rosrun allows you to use the package name to directly run a node within a package (without having to know the package path).
- Usage:

```
$ rosrun [package_name] [node_name]
```

- So now we can run the turtlesim_node in the turtlesim package.
- Then, in a **new tab** 


```
$ rosrun turtlesim turtlesim_node
```

- You will see the turtlesim window:

- **NOTE:** The turtle may look different in your turtlesim window. Don't worry about it - there are many types of turtle and yours is a surprise!

http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

ROS Basics: Computation Graph Level - Practice

- Then, in a **new tab** Ctrl + Shift + T

```
$ rosnode list
```

- You will see something similar to:

```
laptop@mobile-98:~$ rosnode list
/rosout
/turtlesim
```

- One powerful feature of ROS is that you can reassign Names from the command-line.
- Go back to the *rosrun turtlesim* tab and use CTRL + C
- Now let's re-run it, but this time use a Remapping Argument to change the node's name

```
$ rosrun turtlesim turtlesim_node __name:=my_turtle
```

- You will see something similar to:

```
laptop@mobile-98:~$ rosnode list
/my_turtle
/rosout
```

http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

ROS Basics: Computation Graph Level - Practice

- We see our new */my_turtle* node. Let's use another rosnode command, ping, to test that it's up:

```
$ rosnode ping /my_turtle
```

```
laptop@mobile-98:~$ rosnode ping /my_turtle
rosnode: node is [/my_turtle]
pinging /my_turtle with a timeout of 3.0s
xmlrpc reply from http://mobile-98:36039/       time=0.416994ms
xmlrpc reply from http://mobile-98:36039/       time=1.464844ms
xmlrpc reply from http://mobile-98:36039/       time=1.277924ms
xmlrpc reply from http://mobile-98:36039/       time=1.215935ms
xmlrpc reply from http://mobile-98:36039/       time=0.997066ms
xmlrpc reply from http://mobile-98:36039/       time=1.781940ms
```

- Now that you understand how ROS nodes work, let's look at how ROS topics work. Also, feel free to press CTRL + C to stop turtlesim_node.

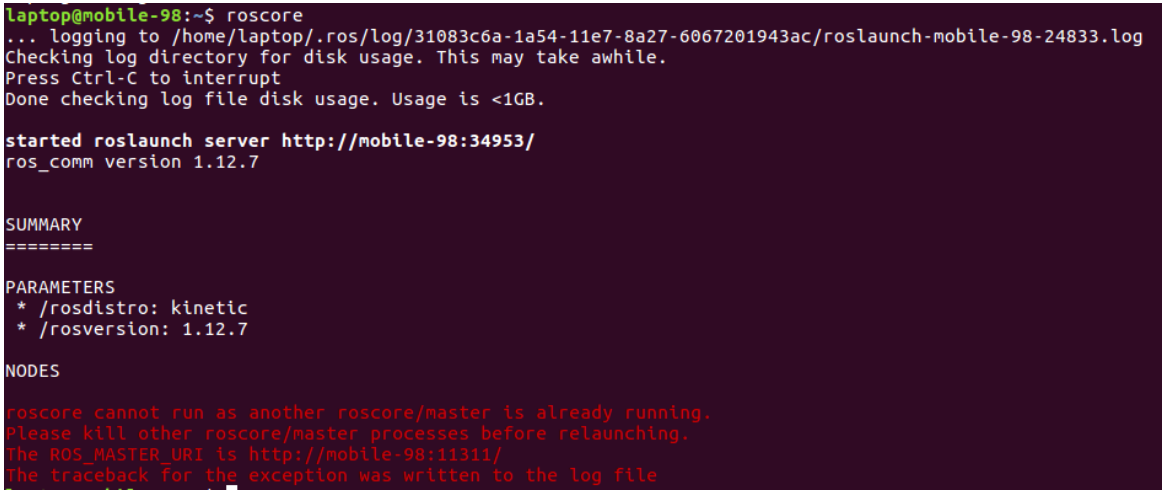
http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

ROS Basics: Computation Graph Level - Practice

- Let's start by making sure that we have roscore running, **in a new terminal**:

```
$ roscore
```

- If you left roscore running from the last tutorial, you may get the error message:



- This is fine. Only one roscore needs to be running.

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROS Basics: Computation Graph Level - Practice

- Now start a new turtlesim_node:

```
$ rosrun turtlesim turtlesim_node
```

- We'll need something to drive the turtle around (**in a new tab**):

```
$ rosrun turtlesim turtle_teleop_key
```

```
laptop@mobile-98:~$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

- Now you can use the arrow keys of the keyboard to drive the turtle around. If you can not drive the turtle **select the terminal window of the turtle_teleop_key** to make sure that the keys that you type are recorded.

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

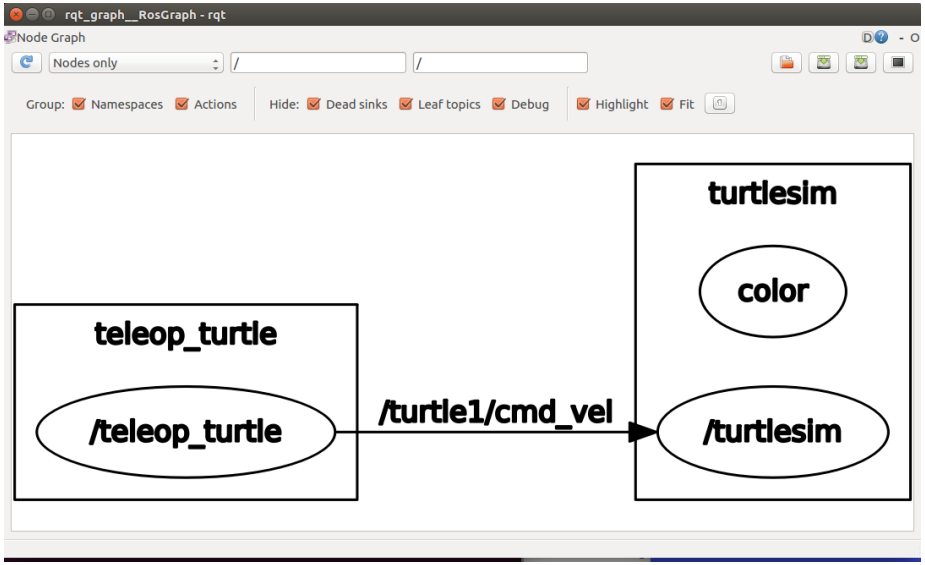
ROS Basics: Computation Graph Level - Practice

- The *turtlesim_node* and the *turtle_teleop_key* node are communicating with each other over a ROS **topic**. *turtle_teleop_key* is **publishing** the key strokes on a topic, while *turtlesim* **subscribes** to the same topic to receive the key strokes. Let's use *rqt_graph* which shows the nodes and topics currently running:

```
$        rosrun rqt_graph rqt_graph
```

- You will see something similar to:

***Note**: If you place your mouse on the topic /turtle1/cmd_vel it will highlight the ROS nodes and the topics*



***Note**: teleop_turtle and turtlesim are communicating over the topic /turtle1/cmd_vel*

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROS Basics: Computation Graph Level - Practice

- The *rostopic* tool allows you to get information about ROS **topics**
- You can use the help option to get the available sub-commands for *rostopic*

```
$        rostopic -h
```

```
laptop@mobile-98:~$ rostopic -h
rostopic is a command-line tool for printing information about ROS Topics.

Commands:
        rostopic bw     display bandwidth used by topic
        rostopic delay  display delay of topic from timestamp in header
        rostopic echo   print messages to screen
        rostopic find   find topics by type
        rostopic hz     display publishing rate of topic
        rostopic info   print information about active topic
        rostopic list   list active topics
        rostopic pub    publish data to topic
        rostopic type   print topic or field type

Type rostopic <command> -h for more detailed usage, e.g. 'rostopic echo -h'
```

- Or double pressing the *tab* key after *rostopic* prints the possible sub-commands:

```
$        rostopic <tab> <tab>
```

```
laptop@mobile-98:~$ rostopic
bw    echo  find  hz    info  list  pub   type
```

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROS Basics: Computation Graph Level - Practice

- *rostopic echo* shows the data published on a topic

```
$ rostopic echo [topic]
```

- Let's look at the command velocity data published by the *turtle_teleop_key node*
  - *This data is published on the /turtle1/cmd_vel topic*

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

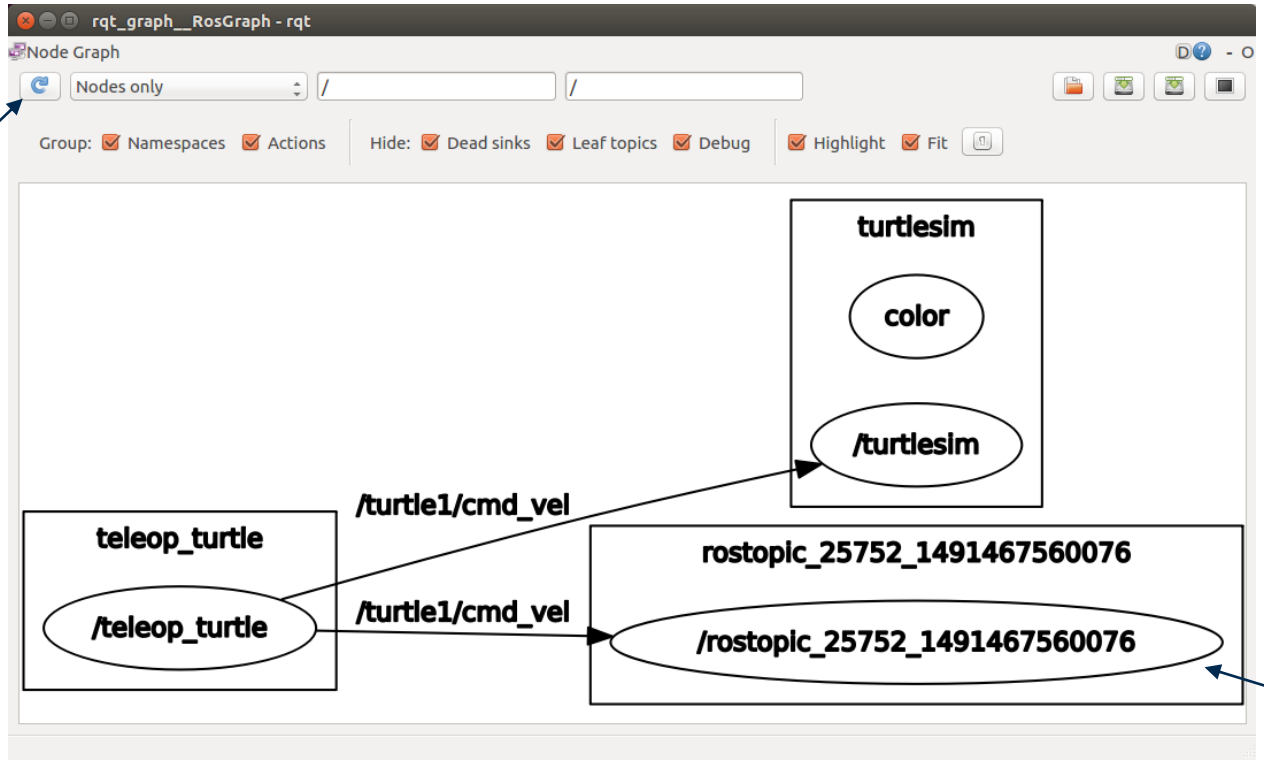
- If you don't see any data published, use the arrow-keys in the *turtle_teleop_key* tab and produce some

```
laptop@mobile-98:~$ rostopic echo /turtle1/cmd_vel
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -2.0
---
```

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROS Basics: Computation Graph Level - Practice

- Now let's look at *rqt_graph again.* Press the refresh button in the upper-left to show the new node. As you can see *rostopic echo*, shown here in red, is now also **subscribed** to the *turtle1/cmd_vel* topic.



***Note**: refresh button*

***Note**: this is rostopic echo*

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROS Basics: Computation Graph Level - Practice

- *rostopic list* returns a list of all topics currently subscribed to and published
- Let's figure out what argument the *list* sub-command needs. In a **new tab** run**:**

```
$ rostopic list -h
```

***Note**: You can use -v or --verbose*

```
laptop@mobile-98:~$ rostopic list -h
Usage: rostopic list [/namespace]

Options:
  -h, --help            show this help message and exit
  -b BAGFILE, --bag=BAGFILE
                        list topics in .bag file
  -v, --verbose         list full details about each topic
  -p                    list only publishers
  -s                    list only subscribers
  --host                group by host name
```

- For *rostopic list* use the verbose option:

```
$ rostopic list -v
```

***Note**: This displays a verbose list of topics to publish to and subscribe to and their type*

```
laptop@mobile-98:~$ rostopic list -v

Published topics:
 * /turtle1/color_sensor [turtlesim/Color] 1 publisher
 * /turtle1/cmd_vel [geometry_msgs/Twist] 1 publisher
 * /rosout [rosgraph_msgs/Log] 3 publishers
 * /rosout_agg [rosgraph_msgs/Log] 1 publisher
 * /turtle1/pose [turtlesim/Pose] 1 publisher

Subscribed topics:
 * /turtle1/cmd_vel [geometry_msgs/Twist] 1 subscriber
 * /rosout [rosgraph_msgs/Log] 1 subscriber
 * /statistics [rosgraph_msgs/TopicStatistics] 1 subscriber
```

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROS Basics: Computation Graph Level - Practice

- Communication on topics happens by sending ROS **messages** between nodes.
- For the publisher (*turtle_teleop_key*) and subscriber (*turtlesim_node*) to communicate, the publisher and subscriber must send and receive the same **type** of message.
- This means that a topic **type** is defined by the message **type** published on it.
- The **type** of the message sent on a topic can be determined using *rostopic type:*

```
$ rostopic type /turtle1/cmd_vel
```

```
laptop@mobile-98:~$ rostopic type /turtle1/cmd_vel
geometry_msgs/Twist
```

- We can look at the details of the message using *rosmsg*:

```
$ rosmsg show geometry_msgs/Twist
```

```
laptop@mobile-98:~$ rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROS Basics: Computation Graph Level - Practice

- *rostopic pub* publishes data on to a topic currently advertised
- Usage:

```
$ rostopic pub [topic] [msg_type] [args]
```

- Example:

```
$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- ‘[2.0, 0.0, 0.0]‘ ‘[0.0, 0.0, 1.8]‘
```

- This will send a single message to turtlesim telling it to move with an linear velocity of 2.0, and an angular velocity of 1.8:

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

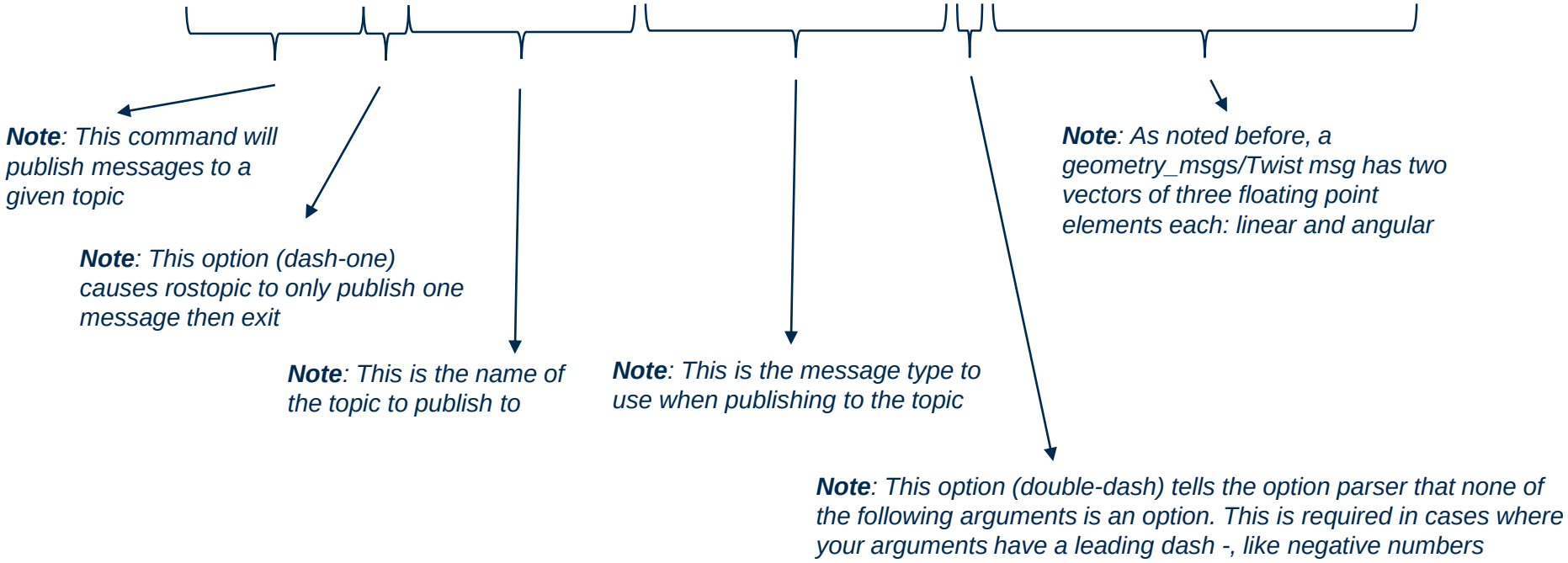
ROS Basics: Computation Graph Level - Practice

- *rostopic pub* publishes data on to a topic currently advertised
- Usage:

```
$ rostopic pub [topic] [msg_type] [args]
```

- Example:

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

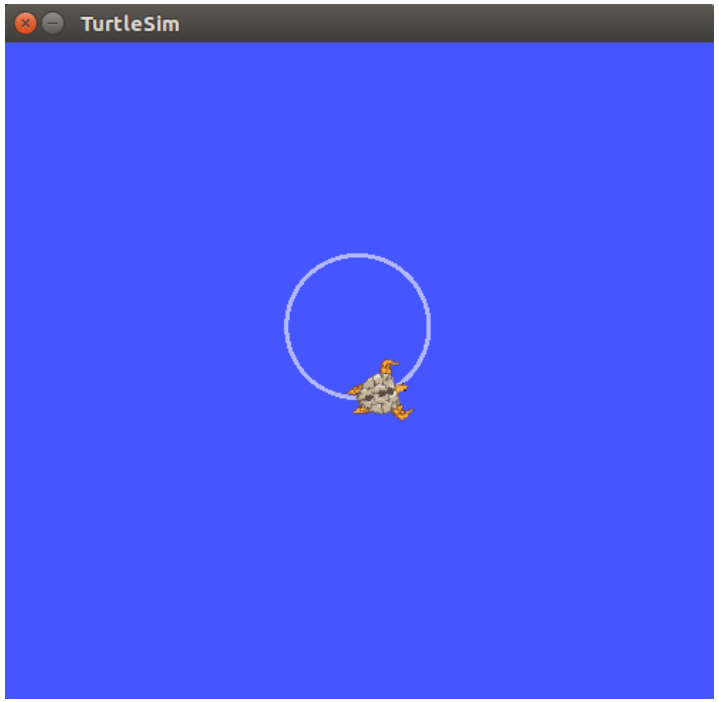


http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROS Basics: Computation Graph Level - Practice

- You may have noticed that the turtle has stopped moving; this is because the turtle requires a steady stream of commands at 1 Hz to keep moving. We can publish a steady stream of commands using *rostopic pub –r* command:

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



http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

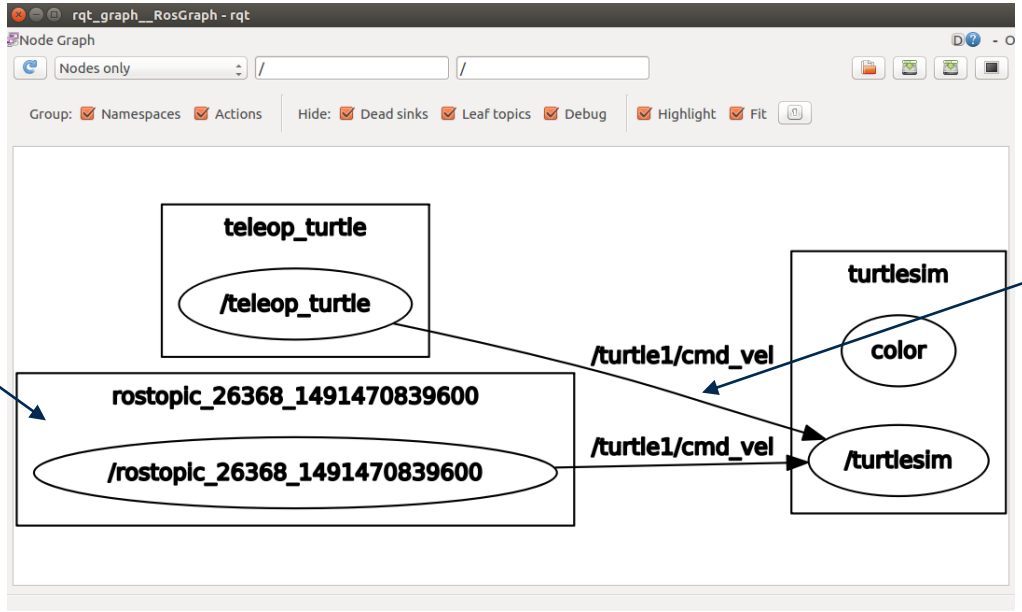
ROS Basics: Computation Graph Level - Practice

- You may have noticed that the turtle has stopped moving; this is because the turtle requires a steady stream of commands at 1 Hz to keep moving. We can publish a steady stream of commands using *rostopic pub –r* command:

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

- We can also have a look at what is happening in *rqt_graph*

***Note**: This is your command line input continously publishing the geometry_msgs/Twist message*



***Note**: Now, two instances are publishing to the same topic*

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROS Basics: Computation Graph Level - Practice

- *rostopic hz* reports the rate at which data is published
- Usage:

```
$ rostopic hz [topic]
```

- Let‘s see how fast the *turtlesim_node* is publishing */turtle1/pose*:

```
$ rostopic hz /turtle1/pose
```

***Note**: turtlesim is publishing data about our turtle at the rate of 60Hz*

```
laptop@mobile-98:~$ rostopic hz /turtle1/pose
subscribed to [/turtle1/pose]
average rate: 62.539
        min: 0.015s max: 0.017s std dev: 0.00041s window: 62
average rate: 62.521
        min: 0.015s max: 0.017s std dev: 0.00040s window: 125
average rate: 62.510
        min: 0.015s max: 0.017s std dev: 0.00044s window: 187
average rate: 62.510
        min: 0.015s max: 0.017s std dev: 0.00043s window: 250
```

- We can also use *rostopic type* in conjunction with *rosmsg show*:

```
$ rostopic type /turtle1/cmd_vel | rosmsg show
```

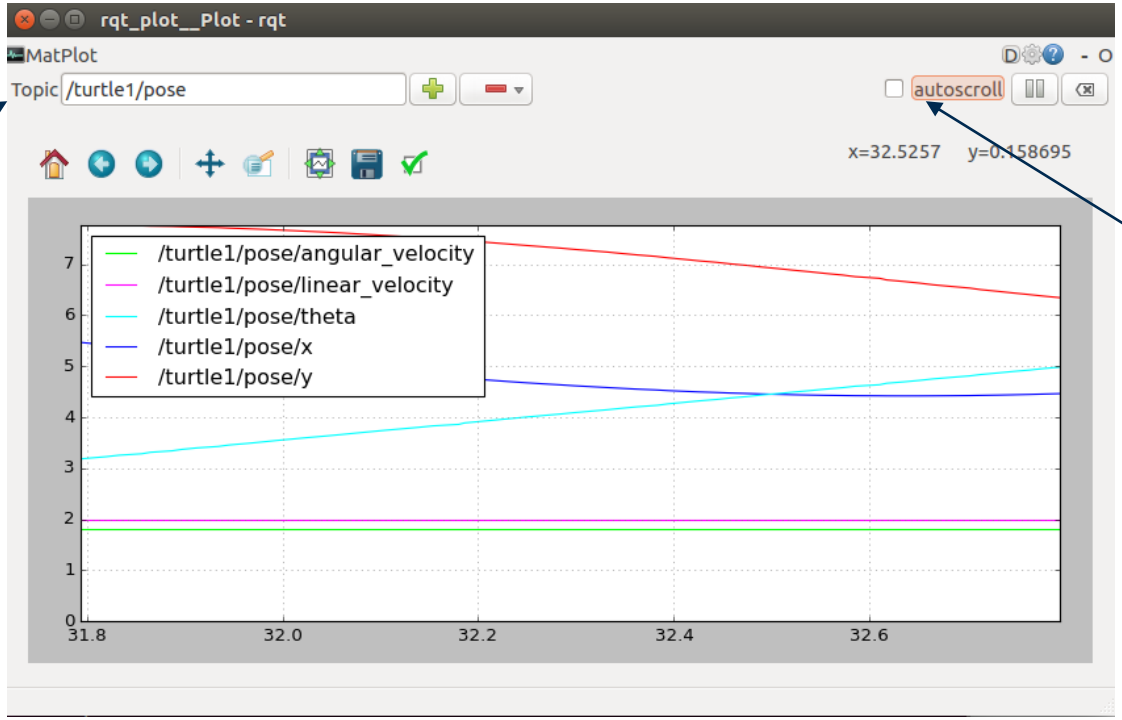
```
laptop@mobile-98:~$ rostopic type /turtle1/cmd_vel | rosmsg show
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROS Basics: Computation Graph Level - Practice

- *rqt_plot* displays a scrolling time plot of the data published on topics.
- Here we'll use *rqt_plot* to plot the data being published on the */turtle1/pose* topic

```
$ rosrun rqt_plot rqt_plot
```



***Note**: Here you have to type in the topic you want to listen to*

***Note**: Pause the record with this checkbox*

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROS Basics: Computation Graph Level - Practice

- *rosservice* can easily attach to ROS's client/service framework with services.
- *rosservice* has many commands that can be used on topics, as shown below:

```
$ rosservice -h
```

```
laptop@mobile-98:~$ rosservice -h
Commands:
        rosservice args print service arguments
        rosservice call call the service with the provided args
        rosservice find find services by service type
        rosservice info print information about service
        rosservice list list active services
        rosservice type print service type
        rosservice uri  print service ROSRPC uri

Type rosservice <command> -h for more detailed usage, e.g. 'rosservice call -h'
```

http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

ROS Basics: Computation Graph Level - Practice

- *rosservice* can easily attach to ROS's client/service framework with services.
- *rosservice* has many commands that can be used on topics, as shown below:

```
$ rosservice -h
```

- The *list* command shows us that the turtlesim node provides nine services:

```
$ rosservice list
```

```
laptop@mobile-98:~$ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/rostopic_26368_1491470839600/get_loggers
/rostopic_26368_1491470839600/set_logger_level
/rqt_gui_py_node_25442/get_loggers
/rqt_gui_py_node_25442/set_logger_level
/rqt_gui_py_node_26984/get_loggers
/rqt_gui_py_node_26984/set_logger_level
/spawn
/teleop_turtle/get_loggers
/teleop_turtle/set_logger_level
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
```

http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

ROS Basics: Computation Graph Level - Practice

- Let's look more closely at the *clear* service using *rosservice type:*

```
$ rosservice type [service]
```

- Let's find out what type the clear service is:

```
$ rosservice type /clear
```

```
laptop@mobile-98:~$ rosservice type /clear
std_srvs/Empty
```

- The service is empty, this means when the service call is made it takes no arguments (i.e. it sends no data when making a **request** and receives no data when receiving a **response**)

http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

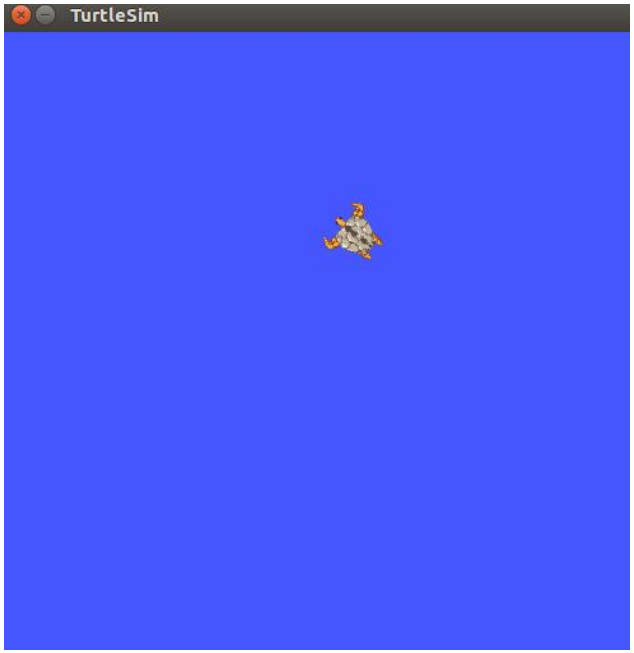
ROS Basics: Computation Graph Level - Practice

- Let's call this service using *rosservice call*

```
$ rosservice call [service] [args]
```

- Here we'll call with no arguments because the service is of the type empty:

```
$ rosservice call /clear
```



http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

ROS Basics: Computation Graph Level - Practice

- Let’s look at the case where the service has arguments by looking at the information for the service spawn

```
$ rosservice type /spawn | rossrv show
```

```
laptop@mobile-98:~$ rosservice type /spawn | rossrv show
float32 x
float32 y
float32 theta
string name
---
string name
```

- This service lets us spawn a new turtle at a given location and orientation. The name field is optional, so let's not give our new turtle a name and let turtlesim create one for us.

```
$ rosservice call /spawn 2 2 0.2 ""
```

```
laptop@mobile-98:~$ rosservice call /spawn 2 2 0.2 ""
name: turtle2
```

http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

ROS Basics: Computation Graph Level - Practice

- This service lets us spawn a new turtle at a given location and orientation. The name field is optional, so let's not give our new turtle a name and let turtlesim create one for us.

```
$ rosservice call /spawn 2 2 0.2 ""
```

http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

ROS Basics: Computation Graph Level - Practice

- *rosparam* allows you to store and manipulate data on the ROS Parameter *Server*
- *rosparam* uses YAML markup language for syntax
- Show the *rosparam* commands:

```
$ rosparam -h
```

```
laptop@mobile-98:~$ rosparam -h
rosparam is a command-line tool for getting, setting, and deleting parameters from the ROS Parameter Server.

Commands:
        rosparam set    set parameter
        rosparam get    get parameter
        rosparam load   load parameters from file
        rosparam dump   dump parameters to file
        rosparam delete delete parameter
        rosparam list   list parameter names
```

- Look at what parameters are currently on the *Parameter Server*:

```
$ rosparam list
```

```
laptop@mobile-98:~$ rosparam list
/background_b
/background_g
/background_r
/rosdistro
/roslaunch/uris/host_mobile_98__45611
/rosversion
/run_id
```

http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

ROS Basics: Computation Graph Level - Practice

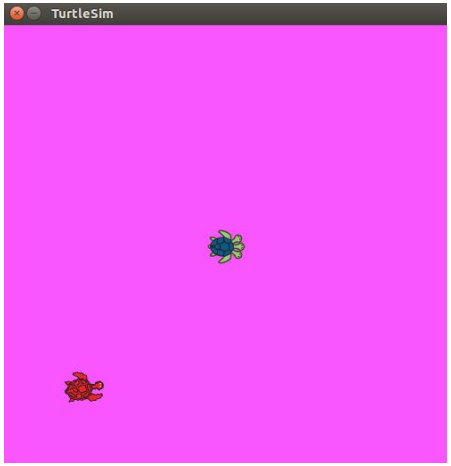
- *rosparam set* and *get* are used for changing and checking parameters
- Change the red channel of the background color:

```
$ rosparam set /background_r 250
```

- This changes the parameter value, now we have to call the clear service for parameter change to take effect:

```
$ rosservice call /clear
```

- Now our turtle environment looks similar to this:



http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

ROS Basics: Computation Graph Level - Practice

- Now let's look at the values of other parameters on the *Parameter Server*
- Get the value of the green channel:

```
$ rosparam get /background_g
```

```
laptop@mobile-98:~$ rosparam get /background_g
86
```

- We can also use *rosparam get /* to show us the contents of the entire *Parameter Server*:

```
$ rosparam get /
```

```
laptop@mobile-98:~$ rosparam get /
background_b: 255
background_g: 86
background_r: 250
rosdistro: 'kinetic

  '
roslaunch:
  uris: {host_mobile_98__45611: 'http://mobile-98:45611/'}
rosversion: '1.12.7

  '
run_id: 20e29326-1b85-11e7-8a27-6067201943ac
```

***Note**: This color channel has been changed by us*

http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

ROS Basics: Computation Graph Level - Practice

- Paramters can also be saved and loaded from files
- Usage:

```
$ rosparam dump [file_name] [namespace]
```

```
$ rosparam load [file_name] [namespace]
```

- Here we write all the parameters to the file params.yaml

```
$ rosparam dump params.yaml
```

- You can have a look at the file:

```
$ vim params.yaml
```

***Note**: You can leave the file by typing:*
:q!

```
background_b: 255
background_g: 86
background_r: 250
rosdistro: 'kinetic

  '
roslaunch:
  uris: {host_mobile_98__45611: 'http://mobile-98:45611/'}
rosversion: '1.12.7

  '
run_id: 20e29326-1b85-11e7-8a27-6067201943ac
```

http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

ROS Basics: Computation Graph Level - Practice

- You can even load these yaml files into new namespaces, e.g. *copy*:

```
$ rosparam load params.yaml copy
```

- Get all parameters in the *copy*-namespace:

```
$ rosparam get /copy/
```

```
laptop@mobile-98:~$ rosparam get /copy/
background_b: 255
background_g: 86
background_r: 250
rosdistro: 'kinetic

  '
roslaunch:
  uris: {host_mobile_98__45611: 'http://mobile-98:45611/'}
rosversion: '1.12.7

  '
run_id: 20e29326-1b85-11e7-8a27-6067201943ac
```

http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

There are much more to be done, if you are interested in.

http://wiki.ros.org/

Outline

Summer School Package

- Open the Terminal
- Set Path as **`/catkin_ws/src`**
- Type:

```
$ git clone
https://git.zlw-ima.rwth-aachen.de/hz658832/summer_school_2017_robotics.git
```

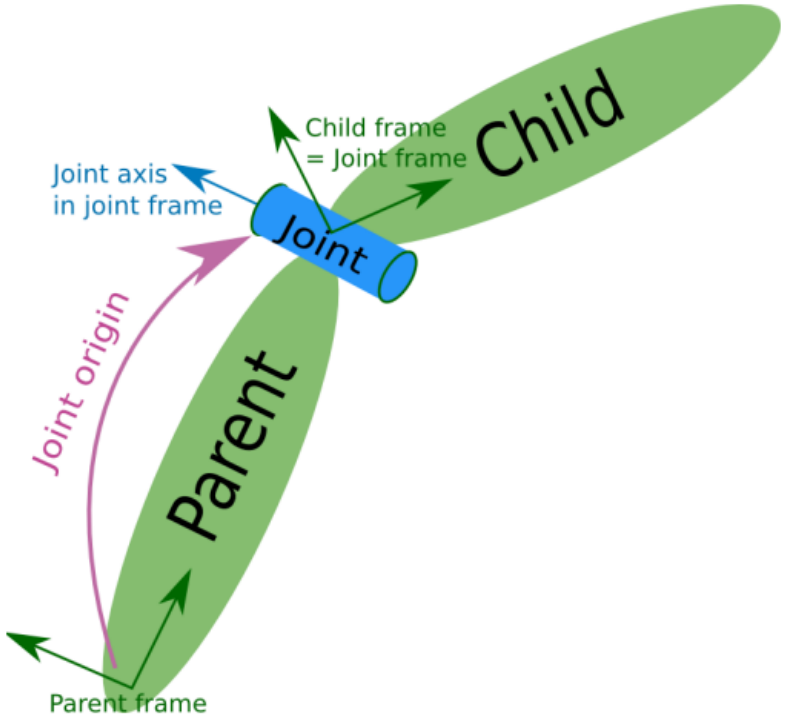
- Go Back to **`/catkin_ws/`** using **`cd ..`**
- Type:

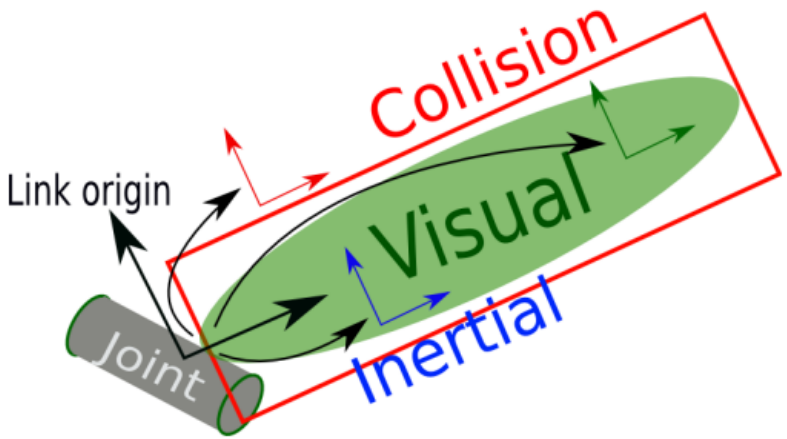
```
$ catkin_make
```

Outline

URDF – Representation of the robot and environment

- URDF file is an XML based description of the environment
- Based on links and joints





URDF – Link element

- Robot link with one frame of reference
- Syntax:
  - name
- Child element *visual*
  - Visual description of the link
  - Geometry primitives (box, cylinder, sphere)
  - Geometry meshes (resources stl/dae)
  - Origin: placement relatively to link reference frame (rpy = fixed axis rotation)
  - material

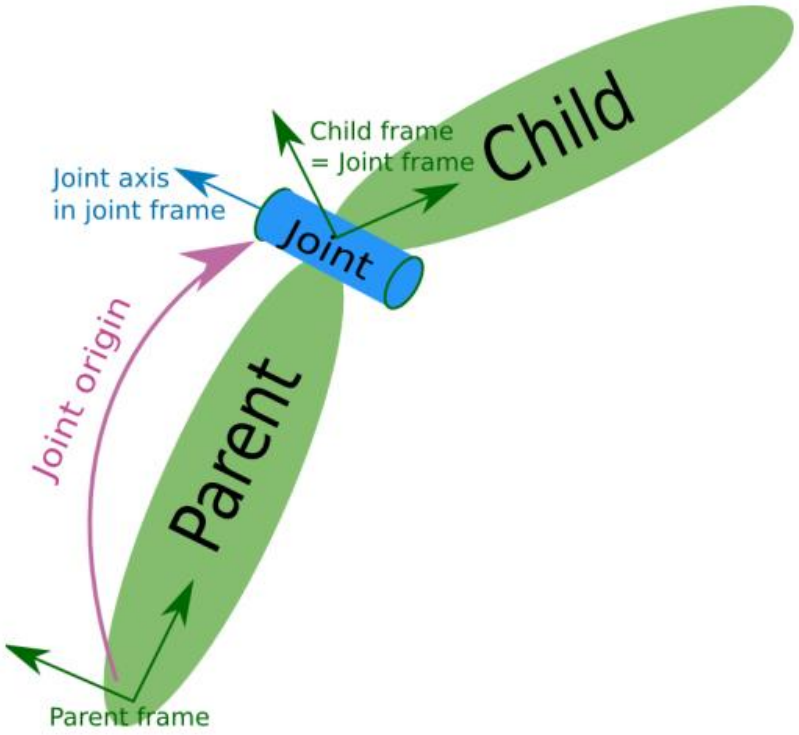
```
<link name="forearm">
 <visual>
  <geometry>
   <origin xyz="0 0 0.1" rpy="0 0 0" />
   <box size="0.1 .2 .5"/>
  </geometry>
  <material name="Cyan">
   <color rgba="0 1.0 1.0 1.0"/>
  </material>
 </visual>
</link>
```

```
<link name="gripper">
 <visual>
  <geometry>
   <mesh filename="package://pkg/m.dae"/>
  </geometry>
 </visual>
 <visual>
  <geometry>
   <cylinder length="0.6" radius="0.2"/>
  </geometry>
 </visual>
</link>
```

URDF – Joint element

```
<joint name="joint1" type="revolute">
 <parent link="forearm"/>
 <child link="gripper"/>
 <origin xyz="0.5 0 0" rpy="0 0 -1.57" />
 <axis xyz="0 0 1" />
</joint>
```

- Robot joint between two links
- Syntax:
  - name
  - type: continuous, fixed, revolute, prismatic, planar, floating
- Child element *parent*
- Child element *child*
- Child element *origin*
  - always in parent reference frame
- Child element *axis*
  - For prismatic and revolute
  - In local joint reference frame



XACRO – A macro language for URDF simplification
A better Way to describe your Robot

- What
  - XML Macro language used for URDF simplification
  - Increase modularity
  - Reduce redundancy
  - Permit parametrization
  - Generates URDF on-the-fly

- How
  - Inclusion
  - Macros
  - Properties
  - Expansion of all xacro statements
  - Command line and output to stdout

XACRO – A macro language for URDF simplification

- Every xml element starts with *xacro*

- Properties
  - Definition
  - Instantiation
  - String concatenation

```
<xacro:property name="width" value=".2"/>
<cylinder radius="${width}" length=".1"/>

<link name="${robotname}s_leg" />

<cylinder radius="${diam/2}" length=".1"/>
```

- Simple math
  - In variables
  - Nested variables
  - No function

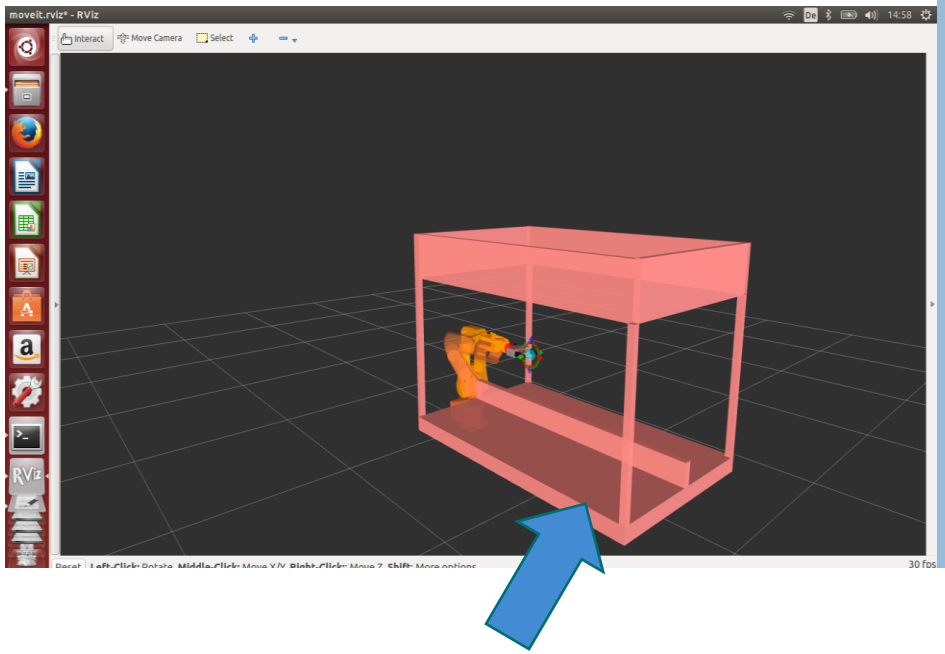
XACRO – A macro language for URDF simplification

- Simple macro
  - Definition
  - Instantiation
- Parametrized macro
  - Definition
  - Instantiation
- Nested macros

```
<xacro:macro name="default_origin">
 <origin xyz="0 0 0" rpy="0 0 0"/>
</xacro:macro>
<xacro:default_origin />
<xacro:macro name="default_inertial" params="mass">
 <inertial>
 <xacro:default_origin />
 <mass value="${mass}" />
  <inertia ixx="0.4" ixy="0.0" ixz="0.0"
           iyy="0.4" iyz="0.0" izz="0.2"/>
  </inertial>
</xacro:macro>
<xacro:default_inertial mass="10"/>
```

XACRO – A macro language for URDF simplification

- An example for the xCell



ground_plane

```
…
<link name="${prefix}ground_plane">
 <visual>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <geometry>
   <box size="1.91 1.19 0.1"/>
  </geometry>
  <xacro:material_abb_gray_white />
 </visual>
 <collision>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <geometry>
   <box size="1.91 1.19 0.1"/>
  </geometry>
  <xacro:material_abb_yellow />
 </collision>
</link>
…
```

Outline

Practicle Unit: Create your first robot model

Kinova

- Plug and Play
- Torque, position, current, temperature and acceleration sensors in each actuator
- Embedded controller
- Torque, position or velocity control
- Intuitive Cartesian control solution provided
- Unlimited joint rotations
- Carbon fiber structure
- Two expansion lines at the end-effector
- Windows/Linux Kinova SDK and ROS enabled

Practicle Unit: Create your first robot model

- Get into

  `/catkin_ws/src/summer_school_2017_robotics/kinova_description/urdf`

- Open **`j2n6s300_full_right.xacro`** with **`Sublime Text`** with

```
$ subl j2n6s300_full_right.xacro  or subl j2n6s300_f (then print Tab)
```

- Your Task:
  - Finish the xacro File with Guide by „ToDo“
  - Write a Macro for „box_link“ and „fixed_joint_link“
  - Build all components within the environment of the Kinova
- Verify your model if you want, using:

```
$ (roscore) | rosrun xacro xacro.py j2n6s300_full.xacro > j2n6s300_full.urdf
```

```
$ check_urdf j2n6s300_full.urdf
```

- Visualize your model if you want, using:

```
$ (roscore) | rosrun xacro xacro.py j2n6s300_full.xacro > j2n6s300_full.urdf
```

```
$ urdf_to_graphiz j2n6s300_full.urdf
```

```
$ (sudo apt install evince) | evince origins.pdf
```

Practicle Unit: Create your first robot model

- If you finished…, let‘s see it in RVIZ

```
$ (roscore) |
  roslaunch kinova_description display_kinova_robot.launch model_name:=“name“
```

Outline

Robot Operating System (ROS)

How to generate motion? => Use Moveit!

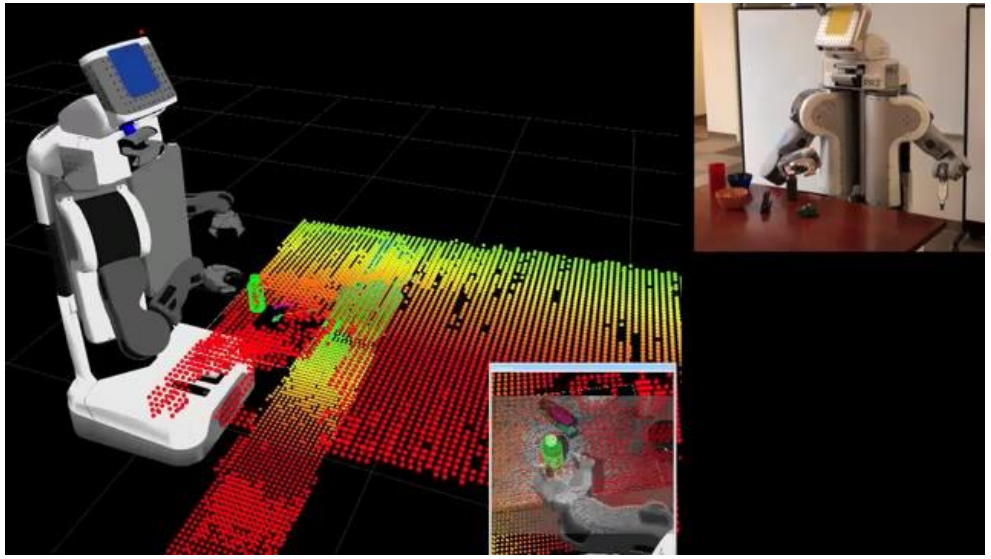
- Moveit is a user-friendly platform for building **flexible** industrial, research and commercial applications
  - Easy configuration, easy programming, quick switch-over
  - High performance

- Features
  - Motion planning
    - Navigation
    - Manipulation
  - Environment integration
    - 3D perception
    - Kinematics
    - control

Robot Operating System (ROS)

What does Moveit offer?

- Technical Capabilities
  - Collision checking: fast and flexible
  - Integrated kinematics
  - Motion Planning
    - Fast, good quality paths
    - Kinematic constraints
  - Standardized interfaces to controllers
  - Execution and monitoring



Outline

Practical Unit: Install MoveIt! and Build your Kinova in it

Install Moveit on your machine

- Install ROS Moveit:

```
sudo apt-get install ros-indigo-moveit-full
```

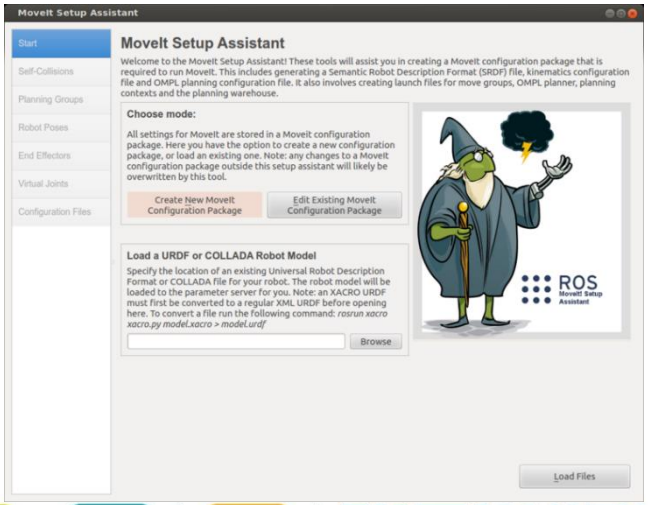
- Restart your PC

```
sudo apt-get update
sudo apt-get dist-upgrade
```

Setup your Kinova

- Run the setup assistant

```
$ (roscore) | roslaunch moveit_setup_assistant setup_assistant.launch
```



- Do it together!
- If you want more information:
  - http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html

Outline

Practical Unit: Make the Kinova move

Motion Planning with Moveit: Try it with your hand

- Run the moveit demo

```
$ (roscore) | roslaunch (your kinova moveit config) demo.launch
```

```
$ (roscore) | roslaunch j2n6s300_moveit_config demo.launch
```

Outline

What is coming…

- Learn Python Programming
- Programming the Motion Planning with Kinova using Python

Thank you for your Attention.