

机器人操作系统

Robot Operating System



东北大学 张云洲

主要内容

- ROS中的基本概念
 - ◆ 定义与体系架构
 - ◆ ROS计算图级概念
 - ◆ ROS文件系统级概念

1 如何看待ROS?

与传统意义上的操作系统不完全相同。

- ROS Wiki对于ROS的定义:

ROS (Robot Operating System) 是一个适用于机器人的开源的元操作系统。它提供了很多操作系统应有的服务，包括硬件抽象，底层设备控制，常用函数的实现，进程间消息传递，以及包管理。它也提供用于获取、编译、编写、和跨计算机运行代码所需的工具和库函数。

1 如何看待ROS?

另一方面——

- ROS不是一个真正意义上的操作系统
- 需要基于其他操作系统才能发挥作用
- 一般来说，ROS需要在Linux（比如Ubuntu）的基础上运行，可以利用ROS中的库，对机器人的导航、通信、路径规划设计算法。

1 如何看待ROS?

ROS 主要目标是为机器人研究和开发提供代码复用的支持。ROS是一个分布式的进程（也就是“节点”）框架，这些进程被封装在易于被分享和发布的程序包和功能包中。

ROS也支持一种类似于代码储存库的联合系统，可以实现工程的协作及发布。这个设计可以使一个工程的开发和实现从文件系统到用户接口完全独立决策（不受ROS限制）。同时，所有的工程都可以被ROS的基础工具整合在一起。

2. ROS的挑战和解决方案

■ 分布式计算

- 一台机器人搭载多台计算机，每台计算机负责部分驱动或传感
- 一台计算机多个进程
- 多个机器人协同完成某复杂任务
- 工作站和机器人之间的交互

■ ROS的解决方案

- 提供了两种机制用于单计算机或多计算机之间不同进程之间的通信

ROS的挑战和解决方案

■ 快速测试

- 机器人调试准备时间长
- 调试过程复杂
- 硬件维护复杂
- 经费有限，设备未就位

■ ROS的解决方案

- 软件模块化设计，底层和上层分开，使用Gazebo等提供高保真仿真，独立测试上层算法
- 提供方便记录实验数据并按时间戳回放的机制，可通过回放数据测试算法，记录的数据格式标准化用于共享。

ROS的挑战和解决方案

■ 软件/代码复用

- 机器人的研发和产业化烟囱式发展严重 (reinvent the wheel)
- 机器人导航、路径规划、建图等通用任务的算法复用率低

■ ROS的解决方案

- 对于通用任务提供ROS标准包
- 提供标准化软硬件通信接口，倒逼机器人软件和硬件标准化

ROS最大的问题

- 目前基于Ubuntu系统，移植嵌入式系统困难
- 无实时性设计
- 体积较大
- 系统整体运行效率低
- **总体上，不适合开发成熟的商业产品。**

3 ROS基本概念

3.1 ROS的系统架构

DDS(Data Distribution Service)

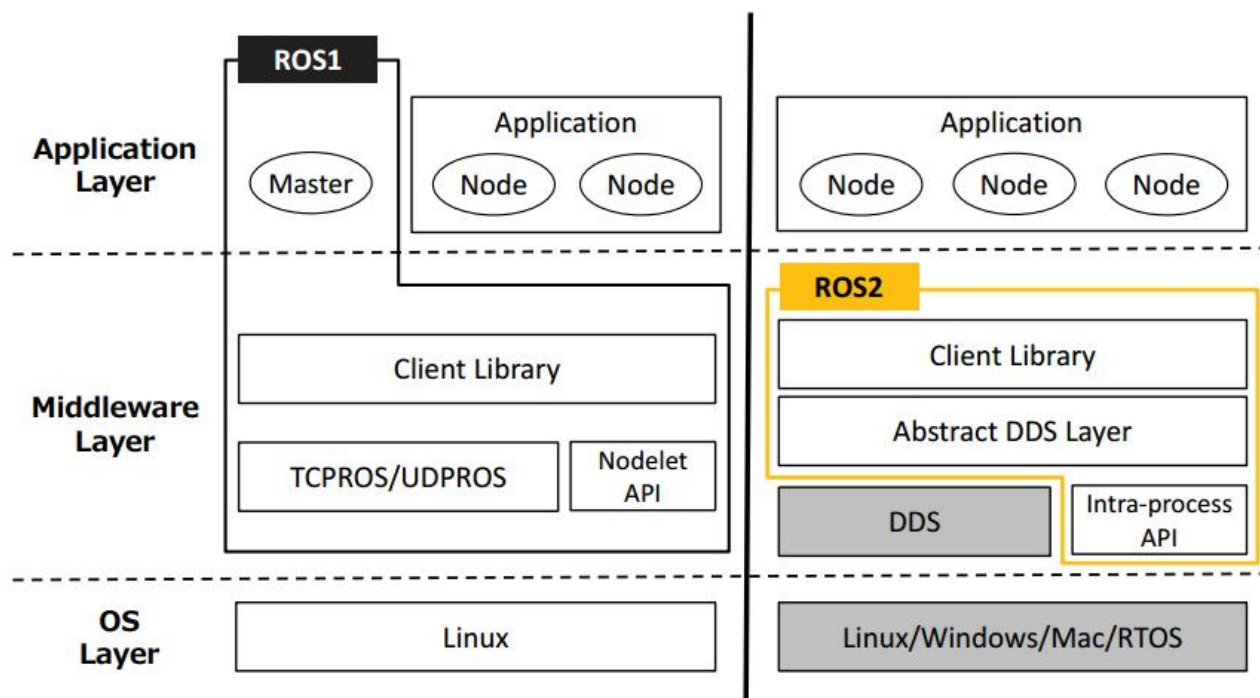
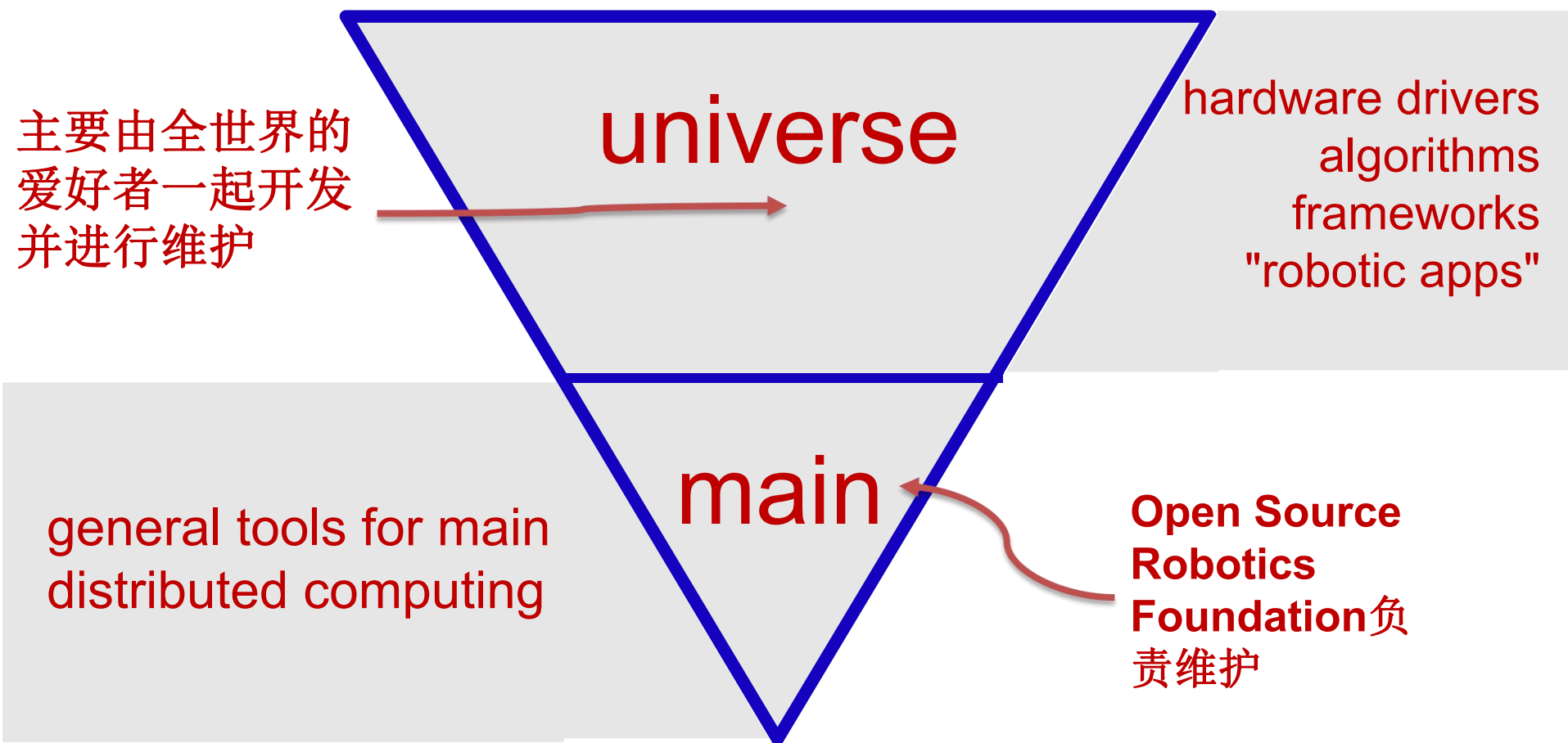
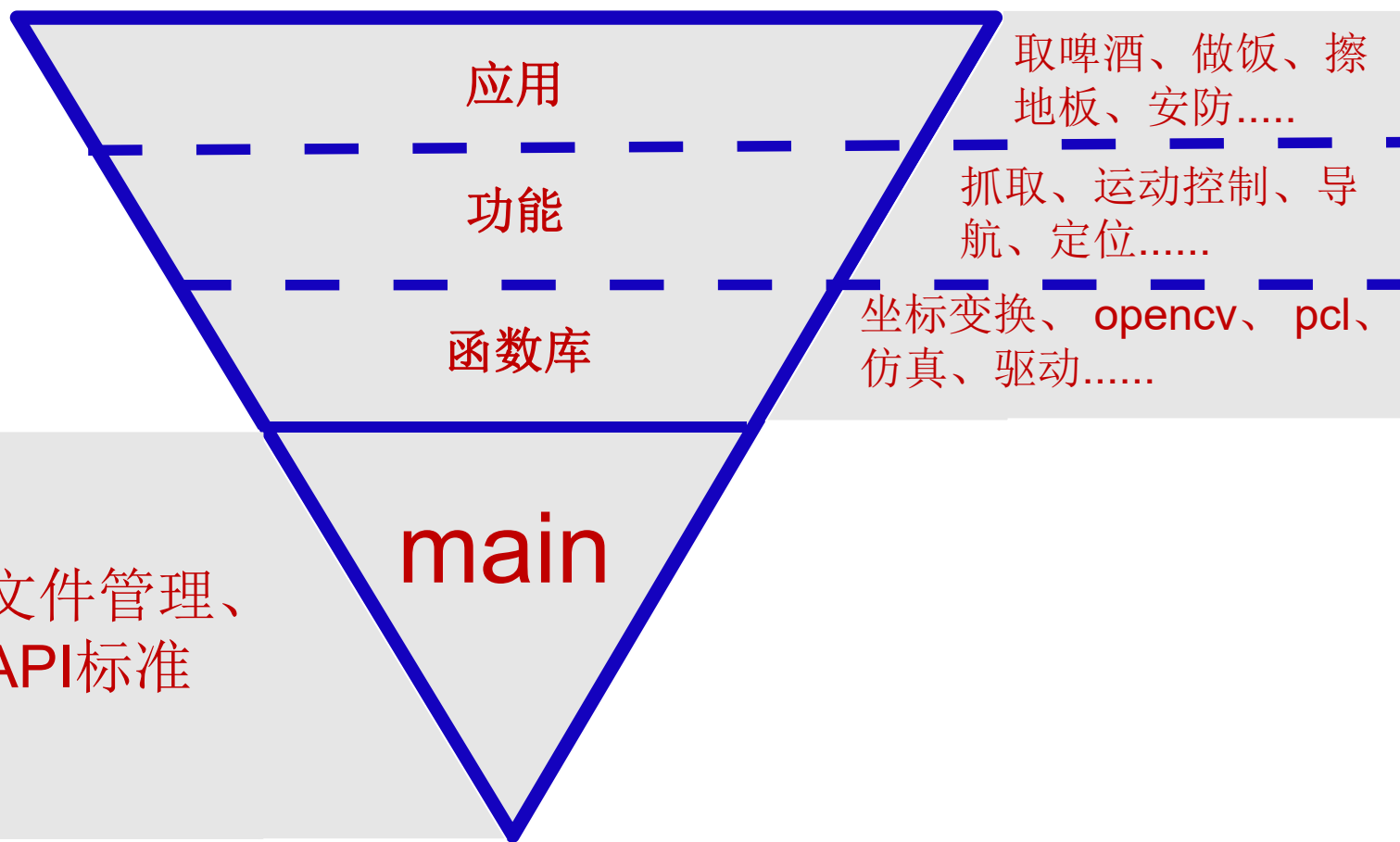


Figure 1: ROS1/ROS2 architecture.

ROS体系结构



ROS体系结构



编译工具、文件管理、
通信架构、API标准

ROS的安装:

- 安装步骤: 参照**ROS Wiki**, 大部分问题可以在网上找答案。

- ROS版本选择:

建议Ubuntu 14.04+ROS Indigo版本或**Ubuntu16.04+Kinetic**;
目前大多数ROS书籍以Indigo为例。

- Ubuntu的每一个版本都有与之对应的ROS, 不能交叉使用。

- 首选使用双系统, 然后再考虑虚拟机。**

- 注意事项: 安装之后进行测试。**

ROS-Ubuntu的版本对应关系

发布日期	ROS版本	对应Ubutnu版本
2016.3	ROS Kinetic Kame	Ubuntu 16.04 (Xenial) / Ubuntu 15.10 (Wily)
2015.3	ROS Jade Turtle	Ubuntu 15.04 (Wily) / Ubuntu LTS 14.04 (Trusty)
2014.7	ROS Indigo Igloo	Ubuntu 14.04 (Trusty)
2013.9	ROS Hydro Medusa	Ubuntu 12.04 LTS (Precise)
2012.12	ROS Groovy Galapagos	Ubuntu 12.04 (Precise)

关于ROS2.0

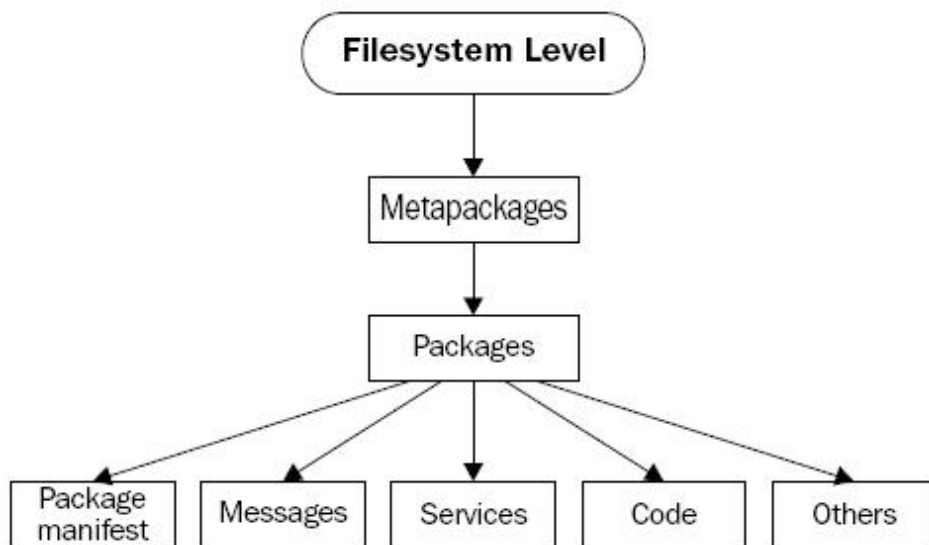
试图解决ROS 1.0中的问题：

- 多个机器人组成的集群——ROS的单master结构
- 小型嵌入式平台，甚至仅仅依赖于微控制器
——ROS依赖于Ubuntu
- 实时系统，包括进程间和跨机器通信——ROS做不到
- ROS网络延迟很大
- ROS可靠性还不够

3 ROS基本概念

3.2 ROS的文件系统级

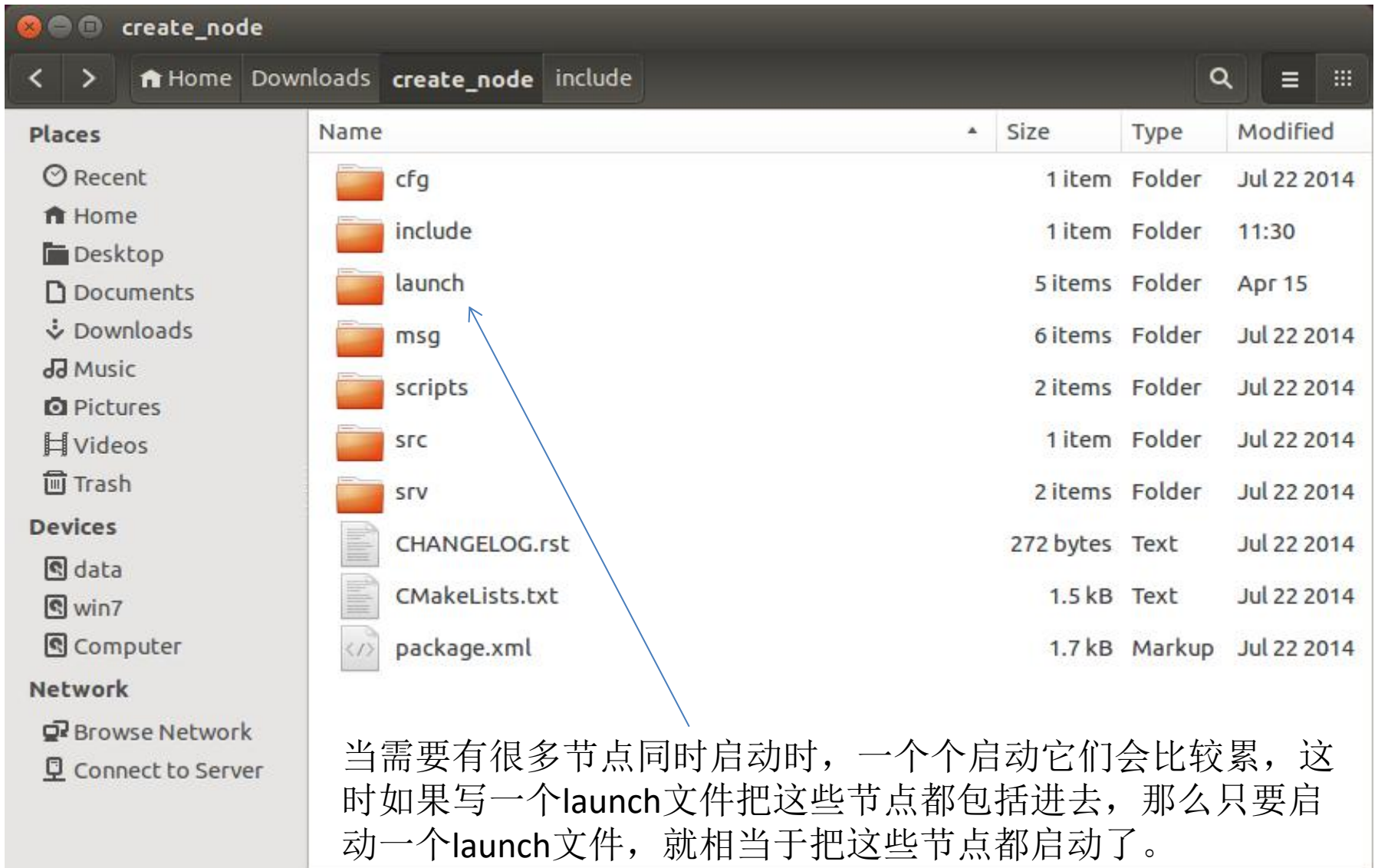
一个ROS程序的不同组件要被放在不同的文件夹下。这些文件夹是根据功能的不同来对文件进行组织的。



```
turtlesim/  
├── CHANGELOG.rst  
├── CMakeLists.txt  
├── images  
│   ├── files  
│   └── hydro.svg  
├── include  
│   └── turtlesim  
│       ├── turtle_frame.h  
│       └── turtle.h  
├── launch  
│   └── multisim.launch  
├── msg  
│   ├── Color.msg  
│   └── Pose.msg  
├── package.xml  
├── src  
│   ├── turtle.cpp  
│   └── turtlesim.cpp  
└── srv  
    ├── Kill.srv  
    └── TeleportRelative.srv
```

TurtleSim功能包

ROS文件系统



The screenshot shows a file manager window titled "create_node" with a sidebar on the left and a main pane on the right. The sidebar contains sections for "Places", "Devices", and "Network". The main pane displays a list of files and folders in a table format. A blue arrow points to the "launch" folder.

Name	Size	Type	Modified
cfg	1 item	Folder	Jul 22 2014
include	1 item	Folder	11:30
launch	5 items	Folder	Apr 15
msg	6 items	Folder	Jul 22 2014
scripts	2 items	Folder	Jul 22 2014
src	1 item	Folder	Jul 22 2014
srv	2 items	Folder	Jul 22 2014
CHANGELOG.rst	272 bytes	Text	Jul 22 2014
CMakeLists.txt	1.5 kB	Text	Jul 22 2014
package.xml	1.7 kB	Markup	Jul 22 2014

当需要有很多节点同时启动时，一个个启动它们会比较累，这时如果写一个launch文件把这些节点都包括进去，那么只要启动一个launch文件，就相当于把这些节点都启动了。

3 ROS基本概念

3.2 ROS的文件系统级

工作空间

一个包含功能包、可编辑源文件或编译包的文件夹。

- 源文件空间
- 编译空间
- 开发空间

```
catkin_ws/  
├── build  
│   ├── catkin  
│   ├── catkin_generated  
│   └── Makefile  
│   └── ...  
├── devel  
│   ├── bin  
│   ├── setup.zsh  
│   └── ...  
└── src  
    ├── CMakeLists.txt -> /opt/ros/hydro/share/catkin/cmake/toplevel.cmake  
    ├── ros_tutorials-hydro-devel  
    └── ...
```

3 ROS基本概念

3.2 ROS的文件系统级

关于CMakeLists.txt:

包集编译时，所有的消息类型、服务类型的C++和Python库都会重新生成，C++代码会重新编译，其它一些在CMakeLists.txt规定的任务都会执行。

当你在包集里新建一个C++文件，请记住；当你在包集里新建一个msg文件时，请记住；当你在包集里新建一个srv文件时，请记住……修改CMakeLists.txt

当你在包集里新建一个Python文件时，不必修改CMakeLists.txt。

3 ROS基本概念

3.2 ROS的文件系统级

功能包 (Package)

在ROS中，所有软件都被组织为软件包的形式，称为ROS软件包或功能包，有时也简称为包。

功能包是ROS中软件组织的基本形式。**一个功能包具有最小的结构和最少的内容，用于创建ROS程序。**它可以包含ROS运行的进程（节点）、配置文件等。

3 ROS基本概念

3.2 ROS的文件系统级

功能包 (Package)

bin/ 编译和链接程序后，存储可执行文件的文件夹。

include/package_name/ 所需要库的头文件。

msg/ 开发非标准消息，请把文件放在这里。

scripts/ 包括Bash、Python或其他的可执行脚本文件。

src/ 存储程序源文件的地方。

srv/ 服务（srv）类型。

CMakeLists.txt CMake的生成文件。

manifest.xml 功能包清单文件。

3 ROS基本概念

3.2 ROS的文件系统级

功能包 (Package)

为了创建、修改或使用功能包，ROS 提供了一些工具：

`rospack` 获取信息或在系统中查找功能包。

`roscat` 查找功能包。

`rosmake` 使用此命令来编译功能包。

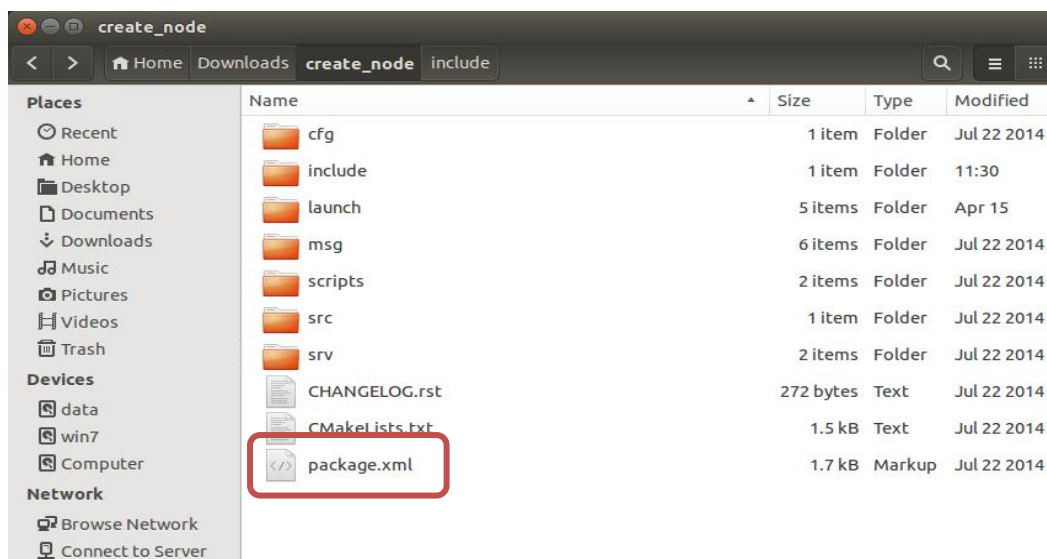
`rosdep` 安装功能包的系统依赖项。

`rqt_deps` 查看功能包的依赖关系图。

3 ROS基本概念

3.2 ROS的文件系统级

- ◆ 功能包元数据文件 (package.xml) : 提供所在功能包的元数据, 包括名字、版本号、功能描述、许可信息、依赖关系等等;



3 ROS基本概念

3.2 ROS的文件系统级

package.xml: 描述功能包的属性, 包括功能包的名字、版本号、作者、维护者、通行证等标签:

<name> - 功能包的名字

<version> - 功能包的版本

<description> - 功能包内容的描述

<maintainer> - 功能包的创建于维护者

<license> - 软件发行版通行证 (例如: GPL, BSD, ASL)

3 ROS基本概念

3.2 ROS的文件系统级

package.xml

- 必须在功能包中，用来说明功能包相关的各类信息。
- 发现在某个文件夹内包含有此文件，那么这个文件夹很可能是一个功能包。

```
?xml version="1.0"?>
<package>
  <name>example</name>
  <version>0.0.1</version>
  <description>
    this is a example.
  </description>
  <maintainer email="test@test.com">test</maintainer>
  <license>BSD</license>

  <url type="website">http://www.test.com</url>
  <author>test</author>

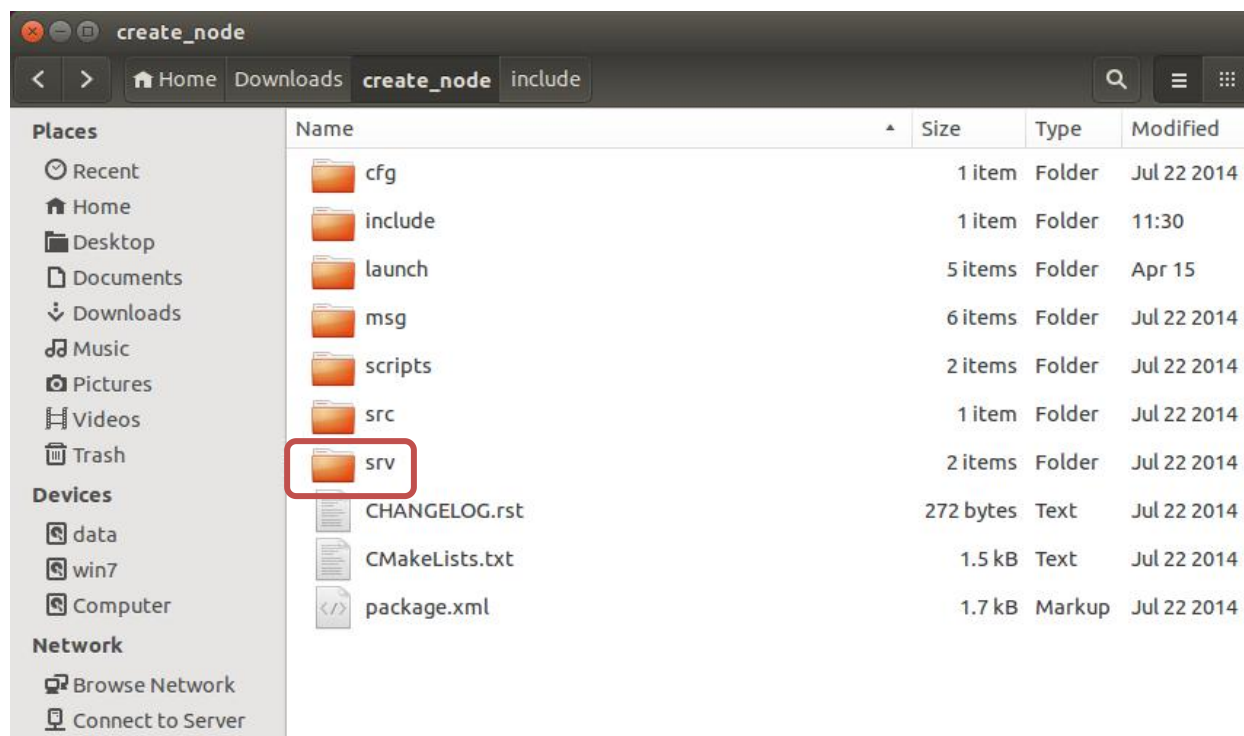
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>geometry_msgs</build_depend>

  <run_depend>geometry_msgs</run_depend>
</package>
```

3 ROS基本概念

3.2 ROS的文件系统级

◆ 服务类型(service types): 对所在功能包定义服务的描述。



3 ROS基本概念

3.2 ROS的文件系统级

- ◆ 元包 (metapackages): 元包表示了一组相关功能包的关系, 对应于某些的功能包集;

3 ROS基本概念

3.2 ROS的文件系统级

消息 (Message)

- ROS使用一种简化的消息类型描述语言来描述 ROS 节点发布的数据值，可使用多种编程语言生成不同类型消息的源代码。
- ROS提供了很多预定义消息类型。创建了一种新的消息类型，需要把消息的类型定义放到功能包的msg/文件夹下。
- 用于定义各种消息的文件，以.msg为扩展名。
- 消息类型必须具有两个主要部分：字段和常量。

字段定义了的消息中传输的数据的类型，例如
int32、float32、string。常量用于定义字段的名称。

```
int32 id  
float32 vel  
string name
```

3 ROS基本概念

ROS消息的标准数据类型

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)
int16	signed 16-bit int	int16_t	int
uint16	unsigned 16-bit int	uint16_t	int
int32	signed 32-bit int	int32_t	int
uint32	unsigned 32-bit int	uint32_t	int
int64	signed 64-bit int	int64_t	long
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	string
time	secs/nsecs signed 32-bit ints	ros::Time	rospy.Time
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration

3 ROS基本概念

报文头:

- ROS消息中的一种特殊数据类型，主要用于添加时间戳、坐标位置等。
- 允许对消息进行编号。通过在报文头内部附加信息，可以知道是哪个节点发出的消息，或者可以添加一些能够被ROS处理的其他功能。

报文头类型包含以下字段:

uint32 seq

time stamp

string frame_id

查看指令: `rosmmsg show std_msgs/Header`

3 ROS基本概念

3.2 ROS的文件系统级

服务 (Service)

- ROS使用一种简化的服务描述语言来描述 ROS 的服务类型。
这直接借鉴了**ROS消息的数据格式**，以实现节点之间的**请求/响应通信**。服务的描述存储在功能包的srv/子目录.srv文件中。
- 调用服务，需要使用该功能包的名称及服务名称。例：
对于sample_package1/srv/sample1.srv文件，可以将它称为sample_package1/sample1服务。
- 在**ROS**中创建一个服务，可以使用服务生成器：只需要在CMakeLists.txt文件中加一行 **gensrv()**命令。

3 ROS基本概念

3.2 ROS的文件系统级

服务 (Service)

srv文件就像msg文件，但包含两部分，请求和响应。这两部分用“---”分开：

int64 A

int64 B

int64 Sum

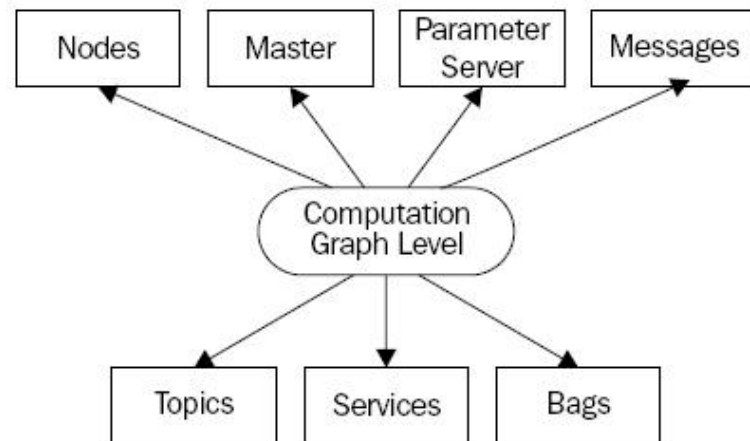
含义：发出请求A和B，然后响应（回应）两个的和sum。

3 ROS基本概念

3.3 ROS的计算图级

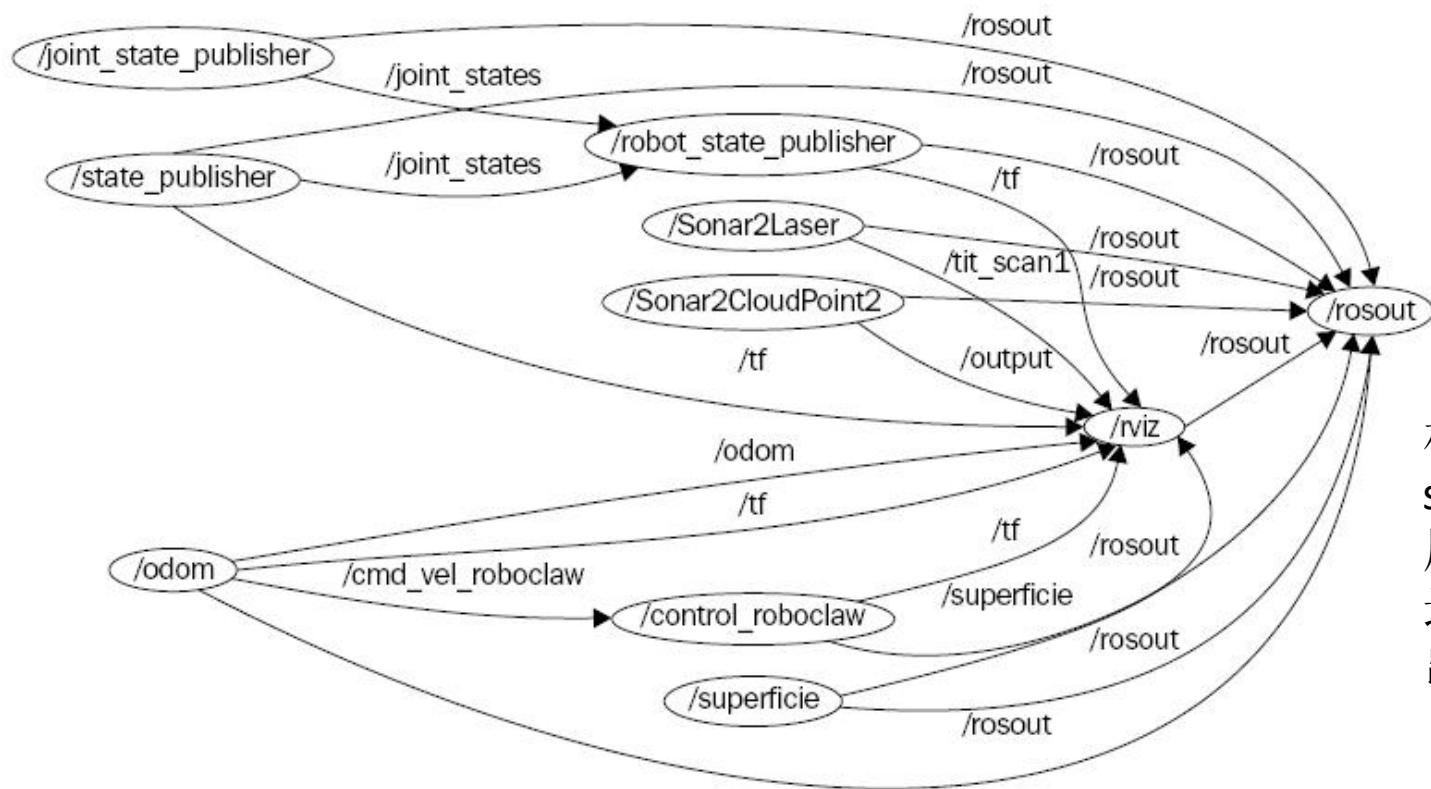
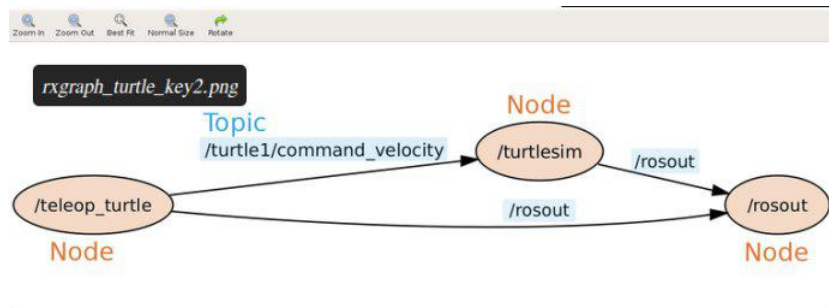
ROS会创建一个连接到所有进程的网络。系统中的任何节点都可以访问此网络，并通过该网络与其他节点交互，获取其他节点发布的信息，并将自身数据发布到网络上。

在这一层级中最基本的概念包括节点、节点管理器、参数服务器、消息、服务、主题和消息记录包，这些概念都以不同的方式向计算图级提供数据。



3 ROS基本概念

3.3 ROS的计算图级：图形化表示



相当于标准的
stdout/stderr;
用于收集和记录
节点调试输出
信息。

3 ROS基本概念

3.3 ROS的计算图级

节点管理器 (master) 与节点 (node)

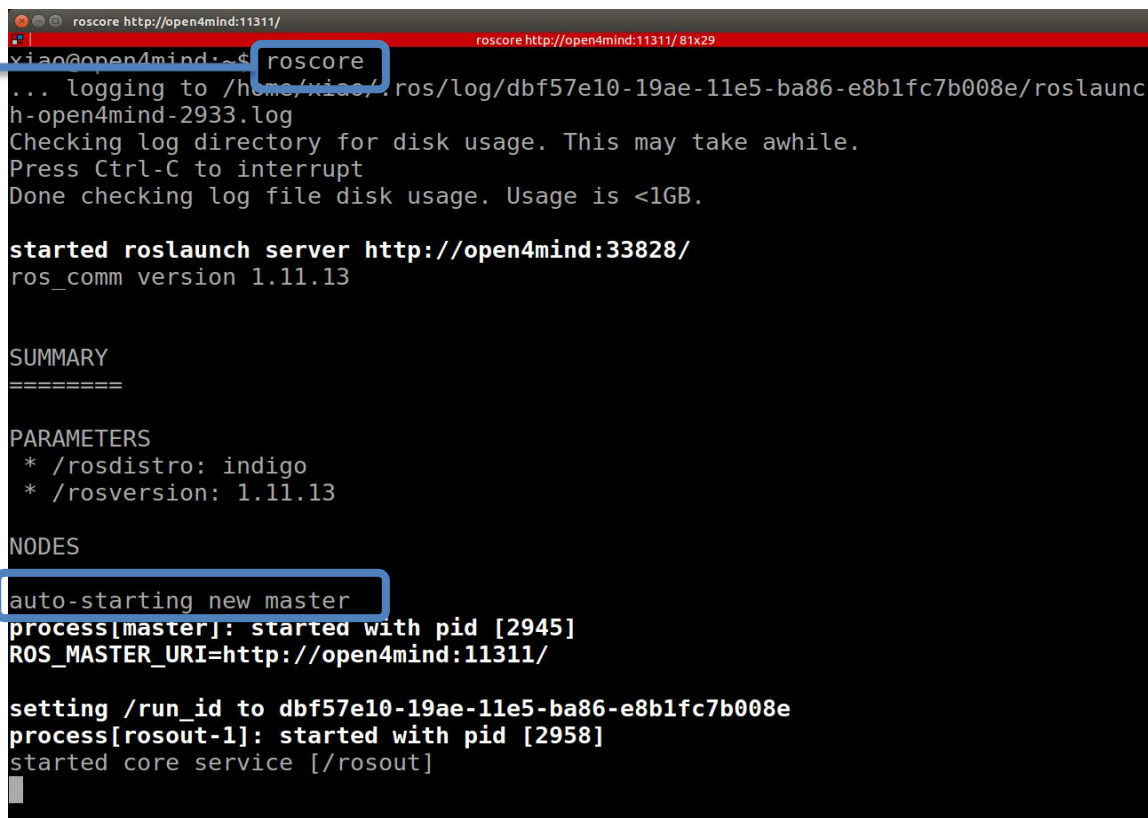
ROS的基本目标是使设计的很多称为节点 (node) 的运行实例能够同时运行。**节点必须能够彼此通信，关键部分就是ROS节点管理器。**

`nodelet`: 特殊节点。在单个进程中运行多个节点，其中每个 `nodelet` 为一个线程 (轻量级线程)。这样可以在不使用ROS网络的情况下与其它节点通信，效率更高且避免网络拥堵。例如，摄像头。

◆主节点/节点管理器（Master）：提供域名注册和查找的功能，类似网络通信中的域名服务器。

~\$ **roscore**

**auto starting
new master**



```
roscore http://open4mind:11311/
xiao@open4mind:~$ roscore
... logging to /home/xiao/.ros/log/dbf57e10-19ae-11e5-ba86-e8b1fc7b008e/roslaunc
h-open4mind-2933.log
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://open4mind:33828/
ros_comm version 1.11.13

SUMMARY
=====

PARAMETERS
* /rostdistro: indigo
* /rosversion: 1.11.13

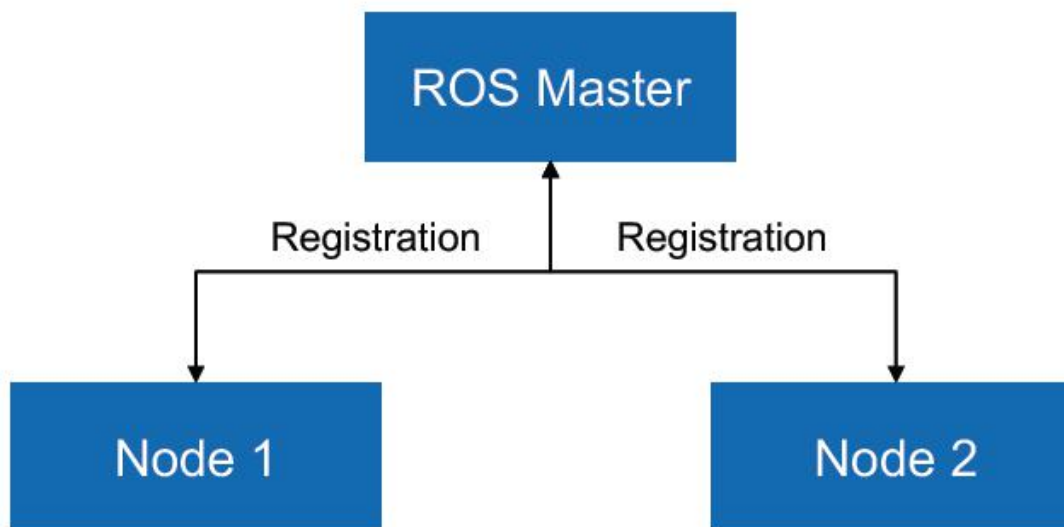
NODES
auto-starting new master
process[master]: started with pid [2945]
ROS_MASTER_URI=http://open4mind:11311/

setting /run_id to dbf57e10-19ae-11e5-ba86-e8b1fc7b008e
process[rosout-1]: started with pid [2958]
started core service [/rosout]
```

3 ROS基本概念

3.3 ROS的计算图级

- ◆节点 (nodes) : ROS以模块化的形式组织计算资源, 节点是用于执行计算任务的进程。



3 ROS基本概念

3.3 ROS的计算图级

ROS程序的核心

- 节点
 - 即可执行文件
 - 由.cpp文件编译生成而来(指C++)
- 节点之间的通讯
 - 节点间信息的发布与接收
 - Publish
 - Subscribe

ROS客户端库允许使用不同编程语言编写的节点之间互相通信:

rospy = python 客户端库

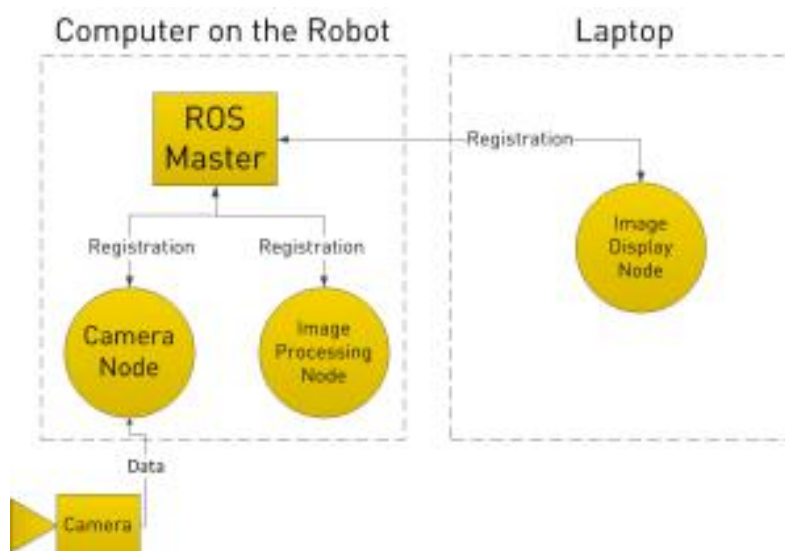
roscpp = c++ 客户端库

节点

- 将整个工程（Project）模块化，每个节点即为一个模块。
- 每个节点都可以单独维护，比如现在有N个节点已经在运行了，用户可以再编写第N+1个节点，然后编译，然后运行，即有N+1个节点共同运行。
- 每个节点有唯一的名字；
- 节点之间可借助网络跨主机共同运行。

举例：

机器人上有个摄像头，然后我们想在机器人和笔记本上看到照片，笔记本就有一个Image display (IDN) 节点，同时机器人上也有节点，一个是Image Processing Node (IRN)， 另一个是Camera Node (CN)。



节点之间的通讯

- 发布者Publisher
- 订阅者Subscriber
- 话题: 发布者与订阅者之间的通讯方式
 - 话题的名称: Topic: 发布者与订阅者之间数据传输“管道”的名字, 为一字符串
 - 消息: Message: 传输数据的类型, 例如int, char, 自定义类型。

ROS节点Node及相关概念的补充

1. 相关概念的补充

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)

ROS节点Node及相关概念的补充

2. 节点Nodes

节点其实就是一个ROS package中的可执行文件，它使用client library与其他Node通信。

nodes可以publish或者subscribe一个Topic，还可以提供或者使用Service。

3. 客户端库 Client Libraries

rospy = python client library

roscpp = c++ client library

5. **roscpp** : 使用ROS时，第一件事情就是运行roscpp。
6. 如果没有initialize roscpp，可能会出现network configuration issue问题。
7. 如果roscpp 没有初始化并且发了一条消息说缺少permissions，则运行：

```
sudo chown -R <your_username> ~/.ros
```

理解ROS节点Node

11. ROS有一个强大的特点是可以通过命令行重新分配名字，如改变节点名字：
`roslaunch turtlesim turtlesim_node __name:=my_turtle`

此时再重新查看`rostopic list` 则会显示：

`/rostopic`

`/my_turtle`

如果名字没变，可能是因为之前用`ctrl+c`关闭了node，尝试：

`rostopic cleanup`

12. 使用节点命令`ping`来测试 `rostopic ping my_turtle`

3 ROS基本概念

3.3 ROS的计算图级

话题 (topic) 和消息 (message)

ROS节点之间进行通信所利用的最重要的机制就是消息传递。在ROS中，消息有组织地存放在话题里。

消息传递的理念是：**当一个节点想要分享信息时，它就会发布 (publish) 消息到对应的一个或者多个话题；当一个节点想要接收信息时，它就会订阅 (subscribe) 它所需要的一个或者多个话题。**

3 ROS基本概念

3.3 ROS的计算图级

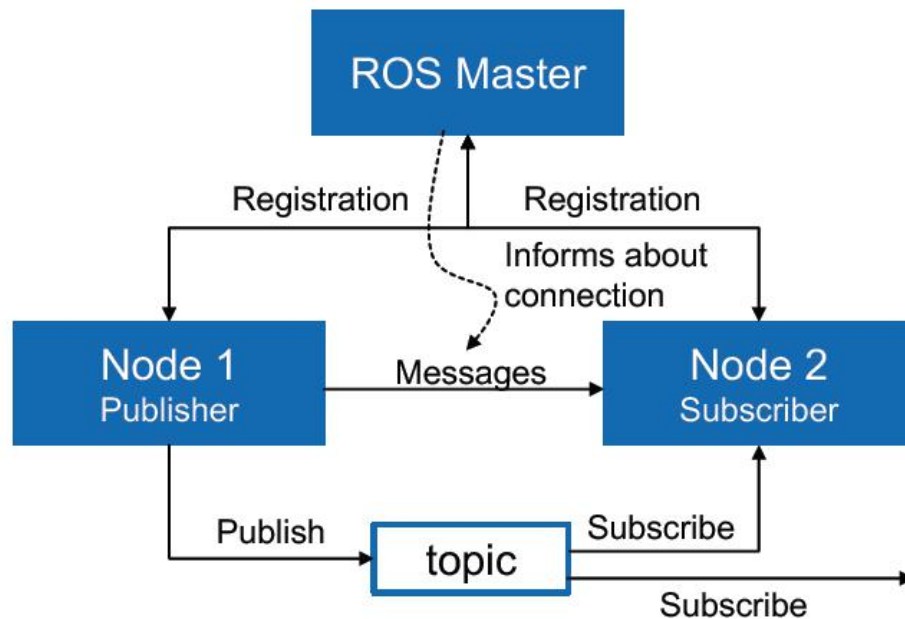
话题 (topic) 和消息 (message)

ROS节点管理器负责确保发布节点和订阅节点能找到对方；
而且消息是**直接地从发布节点传递到订阅节点，中间并不经过节点管理器转交。**

3 ROS基本概念

3.3 ROS的计算图级

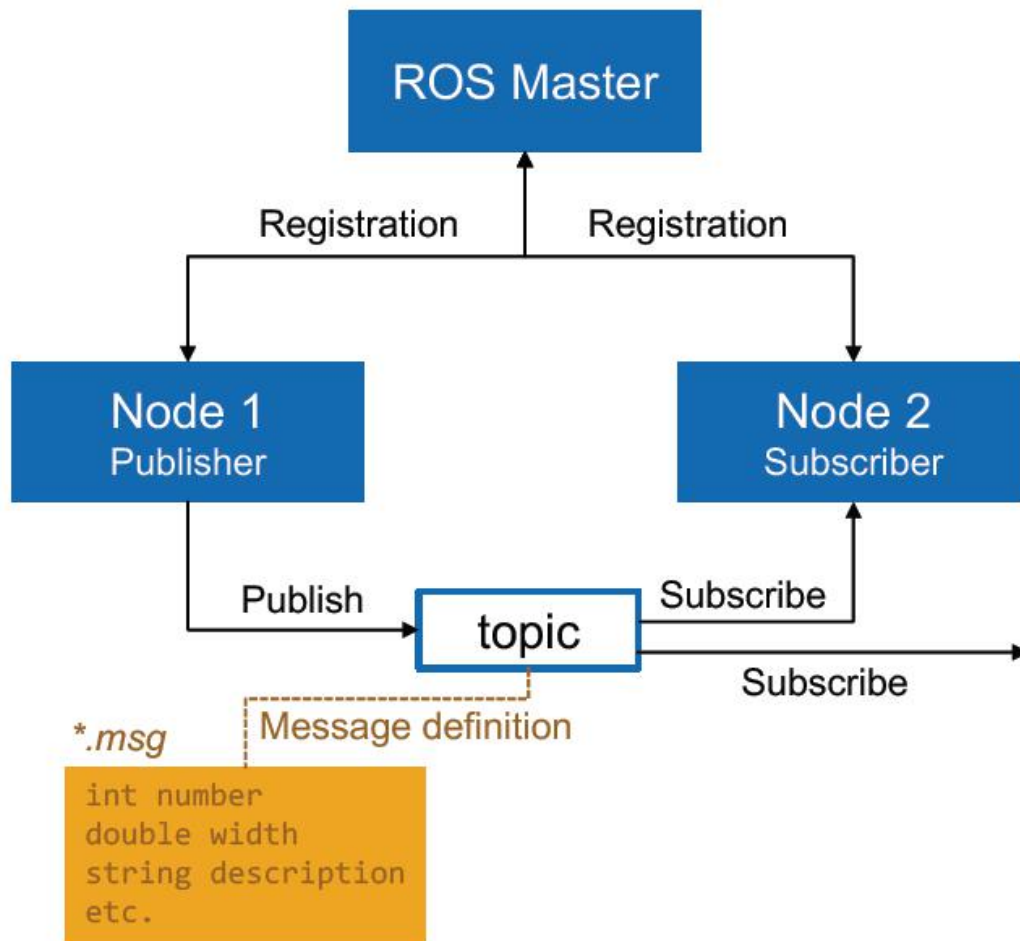
◆主题/话题(topics): 消息是在发布/订阅的框架下传输的, 节点通过发布和订阅相关主题来通信。



3 ROS基本概念

3.3 ROS的计算图级

◆ 消息(messages):
节点之间通过消息进行通信，其实是话题的数据结构。



3 ROS基本概念

3.3 ROS的计算图级

◆消息(messages):

节点之间通过消息进行通信，其实是话题的数据结构。

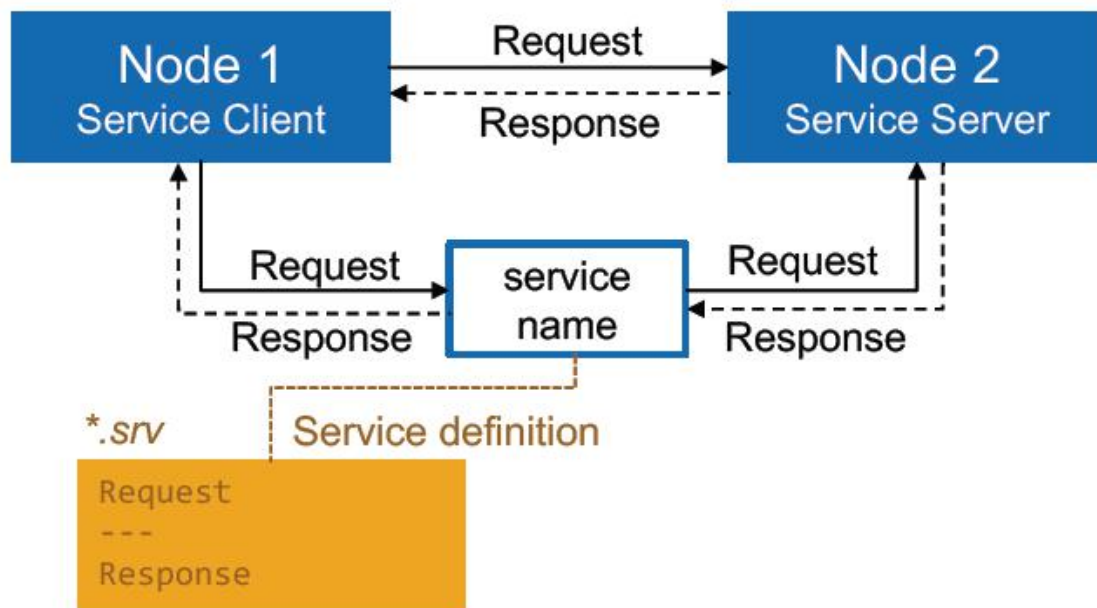
```
Header header  
float32 angle_min  
float32 angle_max  
float32 angle_increment  
float32 time_increment  
float32 scan_time  
float32 range_min  
float32 range_max  
float32[] ranges  
float32[] intensities
```

3 ROS基本概念

3.3 ROS的计算图级

◆服务(services):

发布/订阅可以实现多对多的单向传输，但分布式系统通常有握手的要求，服务即通过请求/应答机制满足这个需求。





ROS:Publisher实例——src/talker.cpp

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>
int main(int argc, char **argv)
{
    ros::init(argc, argv, "talker"); //第三个是节点名字。
    ros::NodeHandle n;

    ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
    //告诉master我们要在chatter话题上发布std_msgs/String的消息。1000，队列大小。
    //NodeHandle::advertise()返回一个ros::Publisher对象，有一个成员函数publish()可发布消息。
    ros::Rate loop_rate(10); //指定循环的频率。10Hz一次循环。

    int count = 0;
    while (ros::ok())
```

```
{  
    std_msgs::String msg;  
  
    std::stringstream ss;  
    ss << "hello world " << count;  
    msg.data = ss.str();  
  
    ROS_INFO("%s", msg.data.c_str()); //printf @ROS  
    chatter_pub.publish(msg); //发布消息  
  
    ros::spinOnce(); //使得回调函数（callback）能够被调用  
    loop_rate.sleep(); //通过sleep休眠来使得发布频率为10hz  
    ++count;  
}  
  
return 0;  
}
```



ROS:Subscriber实例——src/listener.cpp

```
#include "ros/ros.h"
```

```
#include "std_msgs/String.h"
```

```
//回调函数，消息到达chatter topic时会被调用
```

```
void chatterCallback(const std_msgs::String::ConstPtr& msg)
```

```
{   ROS_INFO("I heard: [%s]", msg->data.c_str()); }
```

```
int main(int argc, char **argv)
```

```
{ ros::init(argc, argv, "listener");
```

```
  ros::NodeHandle n;
```

```
  ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback); //订阅消息
```

```
  ros::spin(); //进入自循环，尽可能快的调用消息回调函数。
```

```
  return 0;
```

```
}
```

CMakeLilst.txt文件的更改

//将下列代码加到CMakeLilst.txt

```
include_directories(include ${catkin_INCLUDE_DIRS})
```

```
add_executable(talker src/talker.cpp)
```

```
target_link_libraries(talker ${catkin_LIBRARIES})
```

```
add_dependencies(talker beginner_tutorials_generate_messages_cpp)
```

```
add_executable(listener src/listener.cpp)
```

```
target_link_libraries(listener ${catkin_LIBRARIES})
```

```
add_dependencies(listener beginner_tutorials_generate_messages_cpp)
```

//这样可以加入两个可执行文件，talker和listener，这两个会自动在devel下
//生成，路径是~/catkin_ws/devel/lib/share/

```
add_dependencies(talker beginner_tutorials_generate_messages_cpp)//添加对生  
成的消息文件的依赖
```

```
//运行catkin_make
```

3 ROS基本概念

3.3 ROS的计算图级

■ 参数服务器/数据记录包

- ◆ 参数服务器(parameter server): 参数集中存储处, 是Master的一部分, 允许节点主动查询其感兴趣的参数的值;
- ◆ 数据记录包(bags): 用于记录和重现ROS中的数据。

3 ROS基本概念

3.3 ROS的计算图级

常用命令

```
source /opt/ros/kinetic/setup.bash
```

```
roscore
```

#运行ROS程序之前必须运行的步骤

```
cd ~/catkin_ws #进入工作空间
```

```
catkin_make
```

#编译

```
mkdir -p ~/catkin_ws/src
```

```
cd ~/catkin_ws/
```

```
catkin_make
```

#创建catkin工作空间

```
source devel/setup.bash
```

在catkin的src目录下:

```
catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

#创建新

程序包

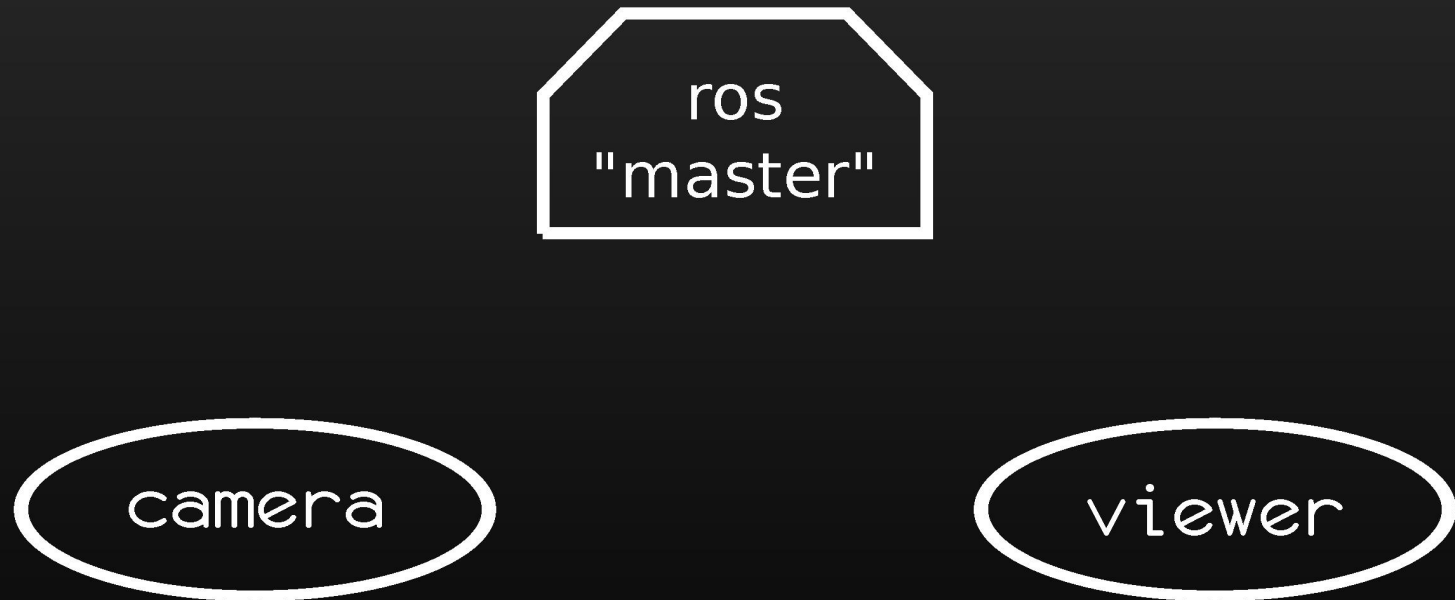
3 ROS基本概念

3.4 ROS的工具

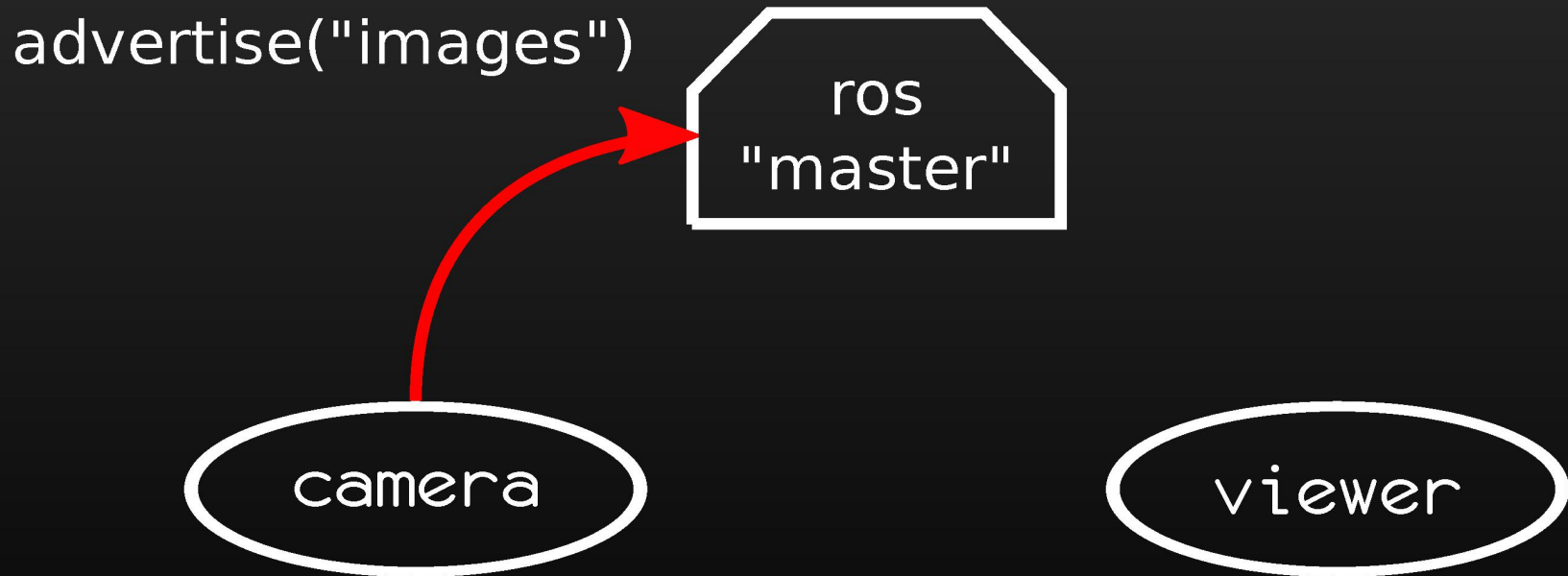
rqt:

rqt是ROS下的一个软件框架，以插件的方式提供了很多方便开发者使用的GUI界面，可以实时查看ROS中流动的消息。

ROS异步通信过程（基于topics）



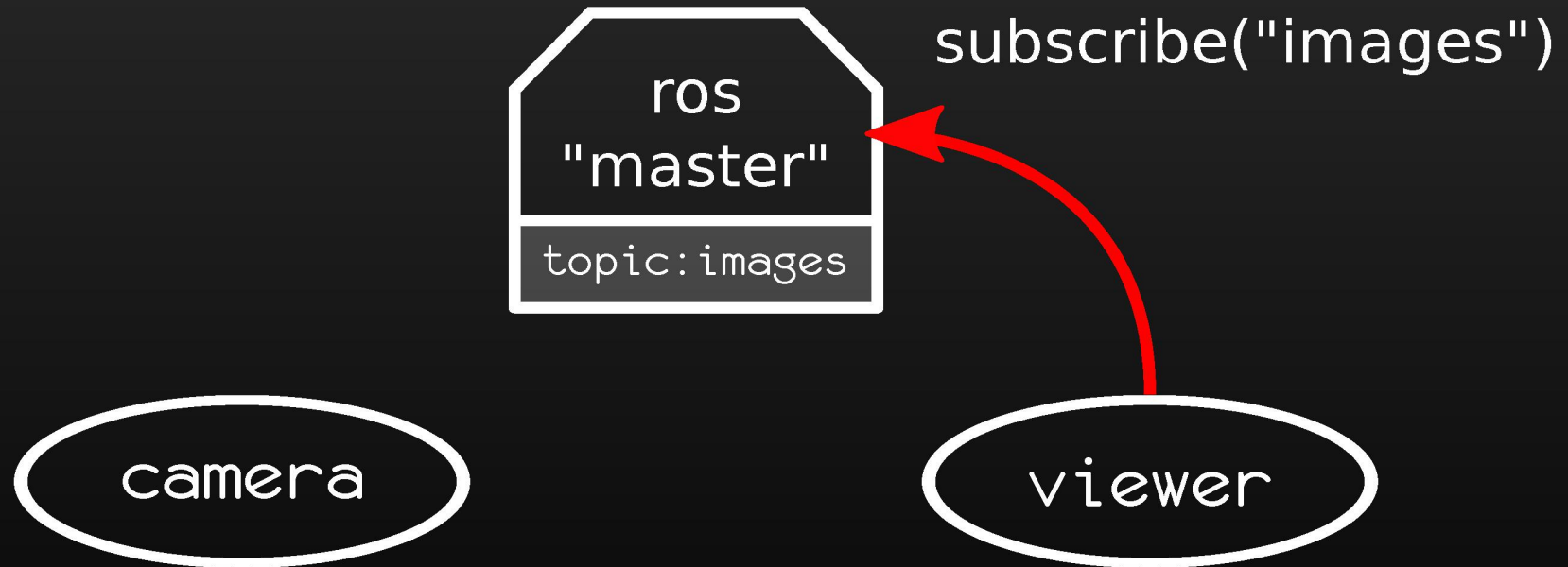
ROS异步通信过程（基于topics）



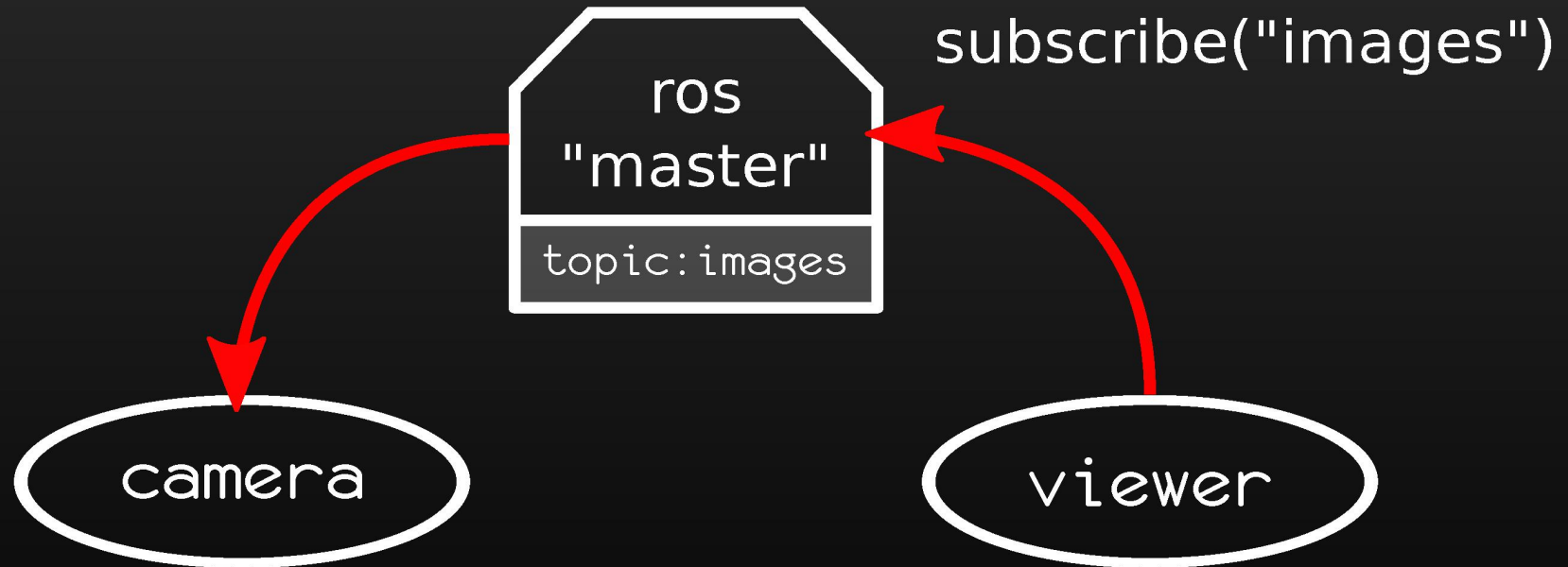
ROS异步通信过程（基于topics）



ROS异步通信过程（基于topics）



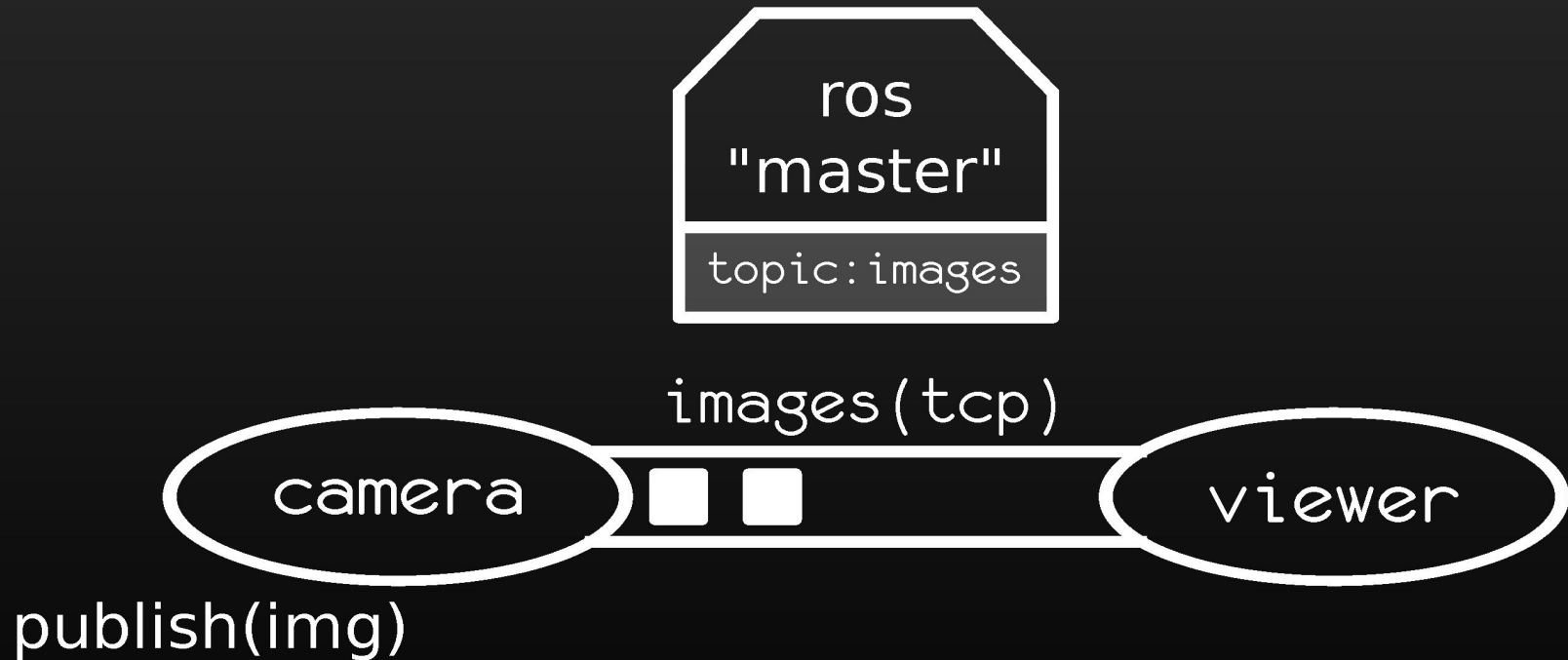
ROS异步通信过程（基于topics）



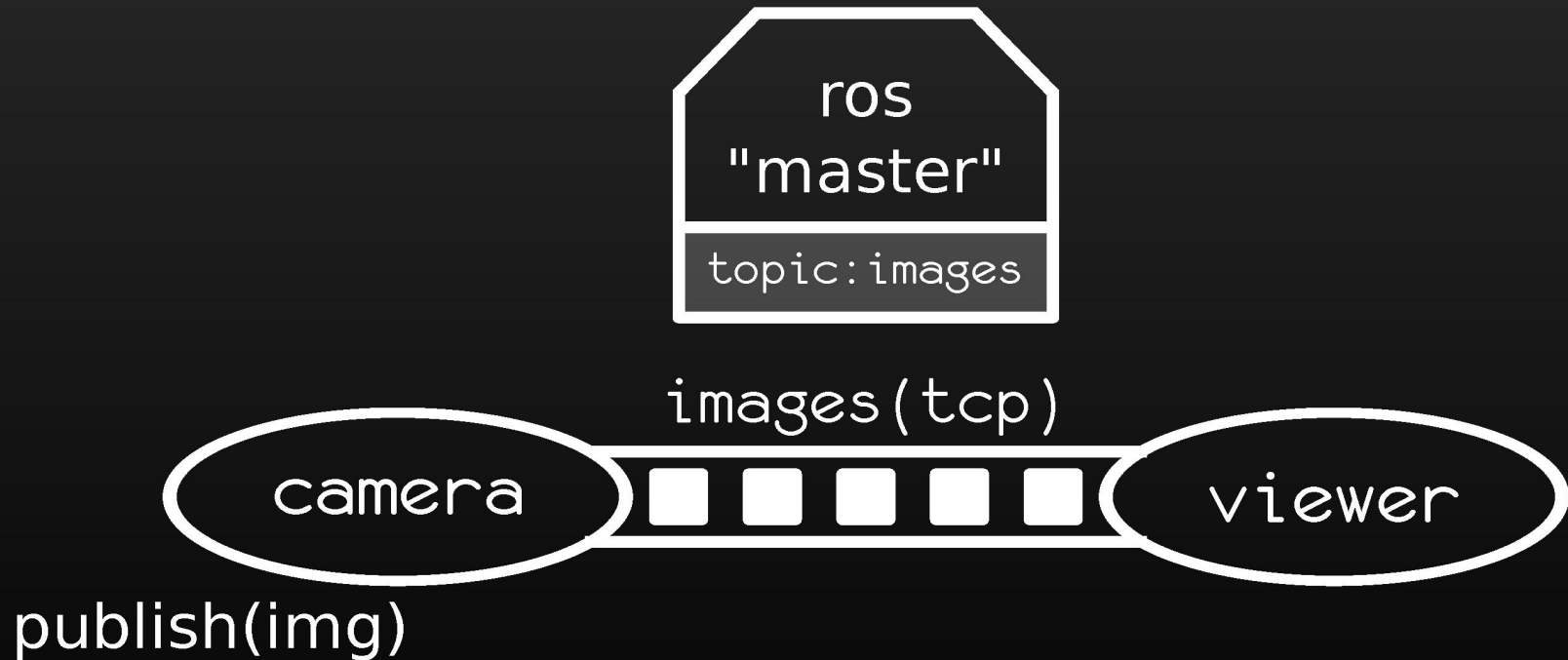
ROS异步通信过程（基于topics）



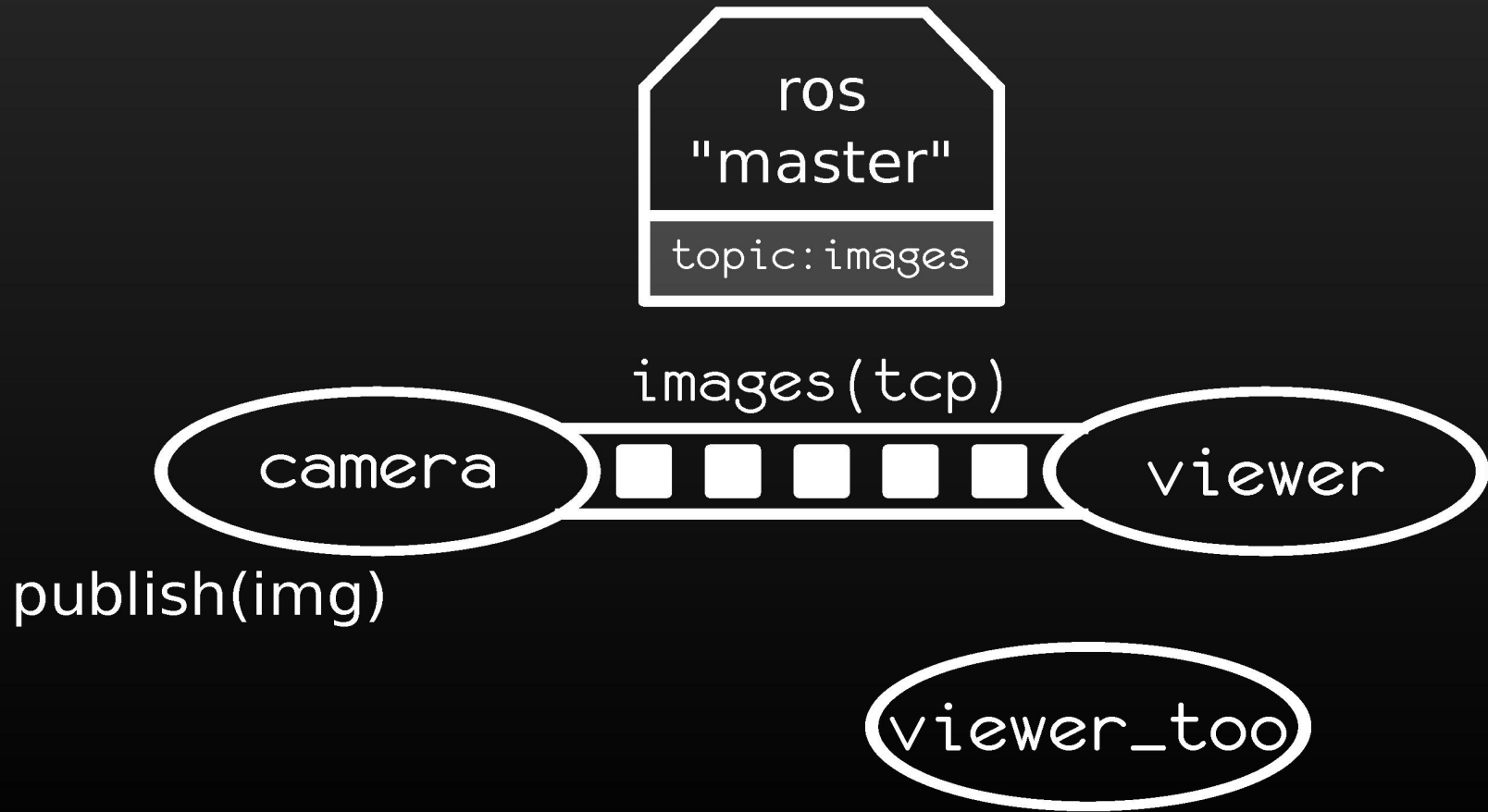
ROS异步通信过程（基于topics）



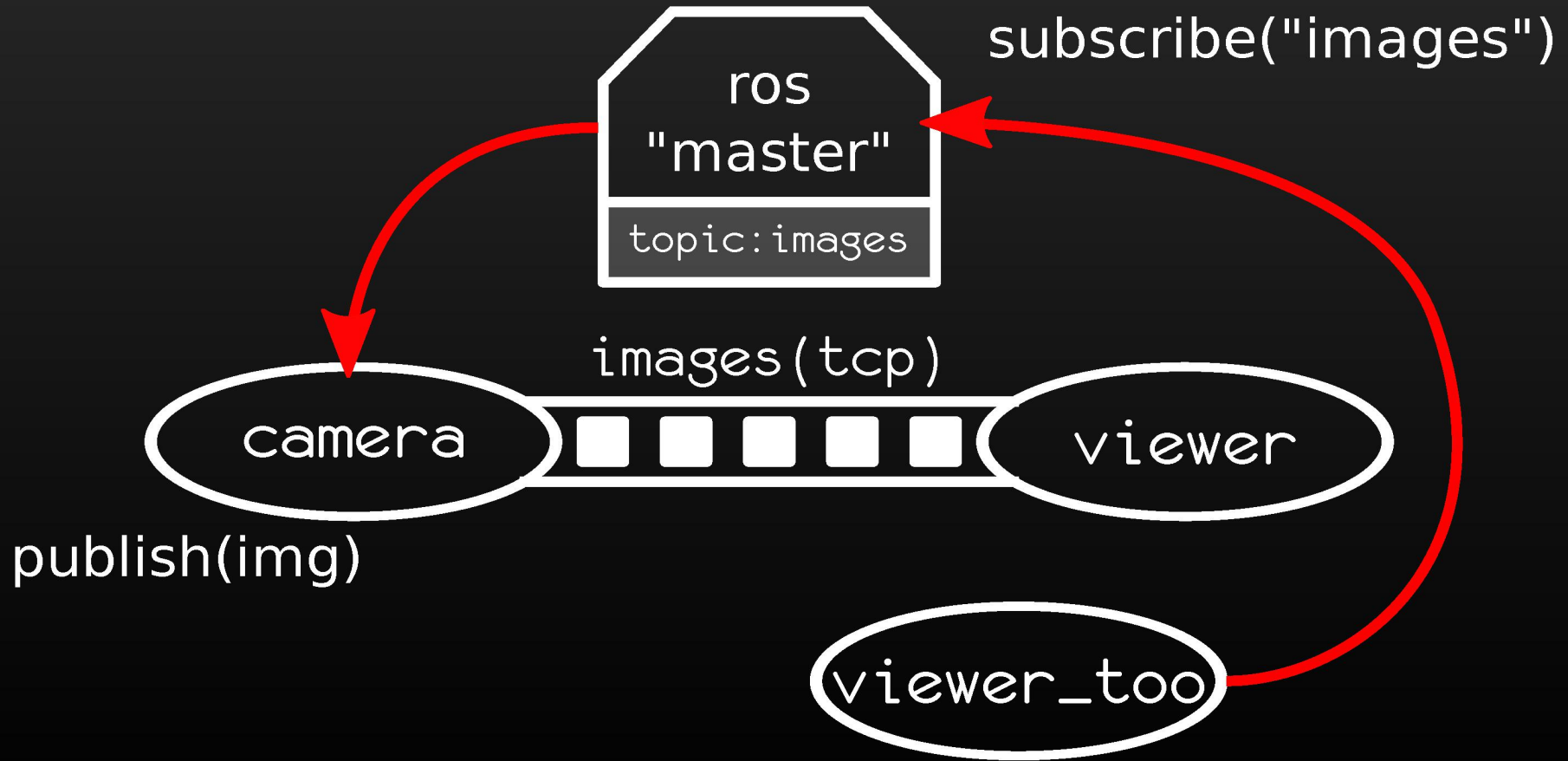
ROS异步通信过程（基于topics）



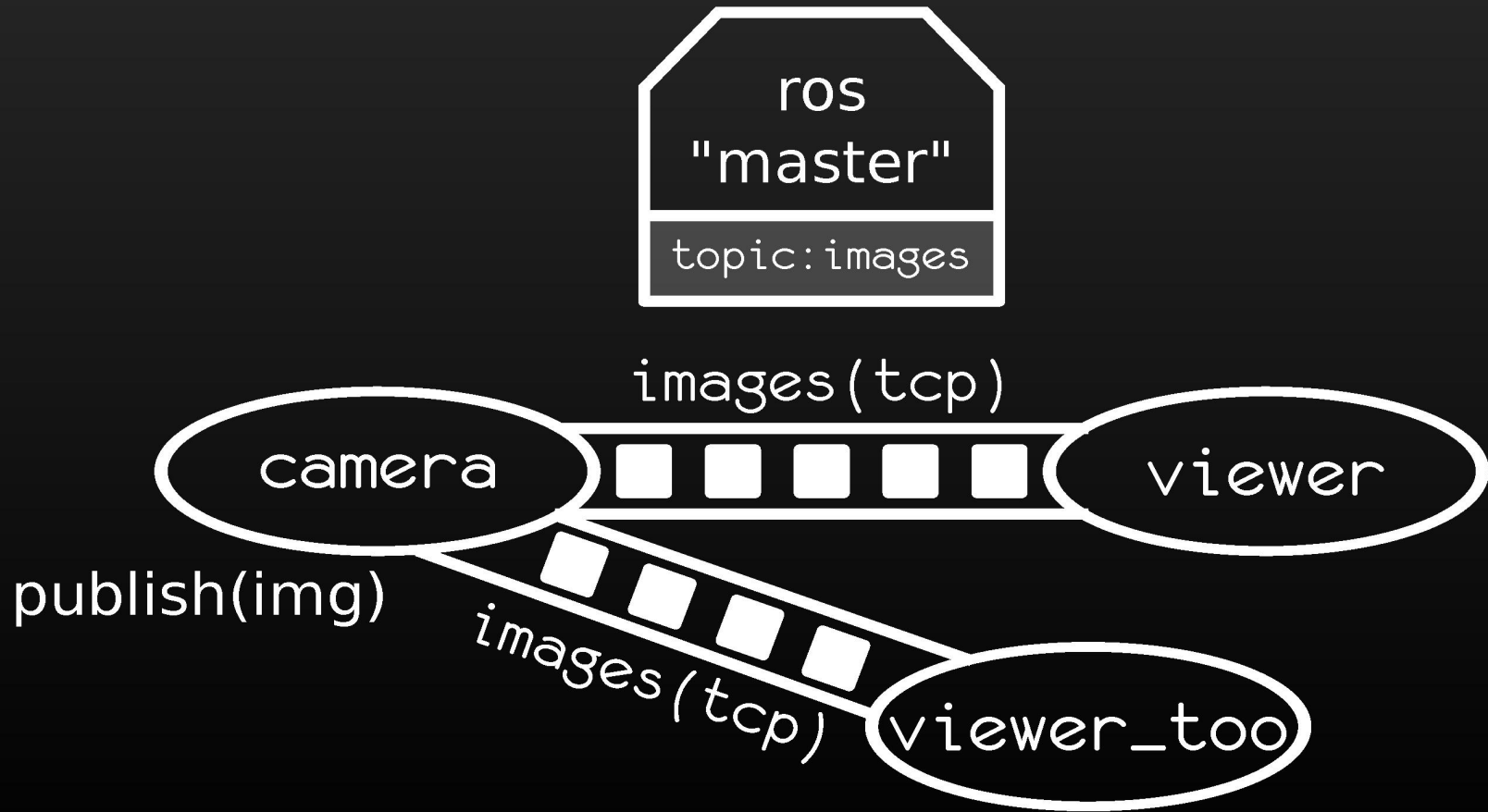
ROS异步通信过程（基于topics）



ROS异步通信过程（基于topics）



ROS异步通信过程（基于topics）



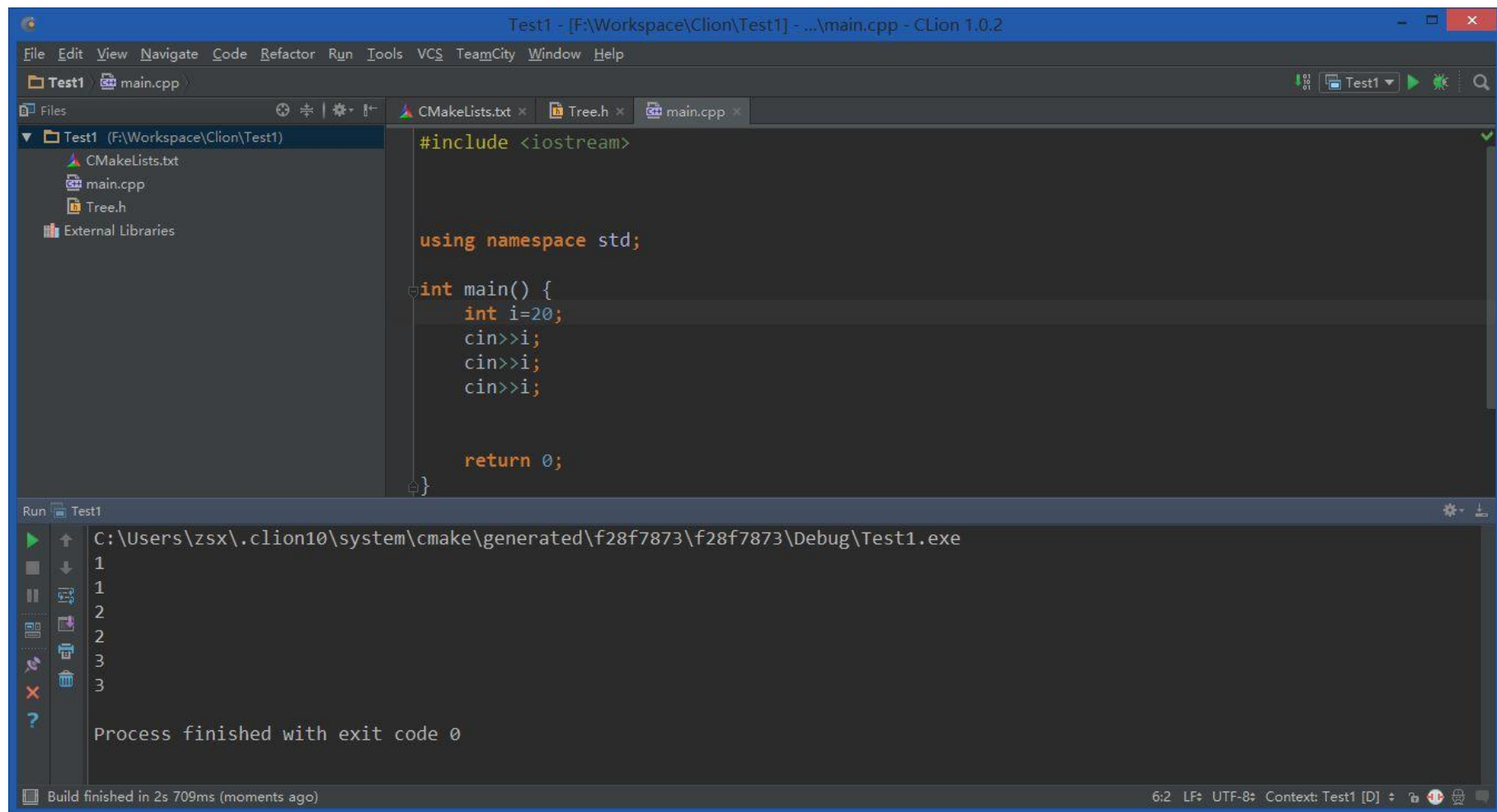
3 ROS基础

3.4 IDE的选择与安装

为什么要使用IDE?

- 使用自带的文本编辑器不方便对程序进行管理
- 调试时需要断点、自动补全、跳转等功能
- 以上提及的IDE一般支持cmake工程，方便程序编译运行
- 对C++支持较好

3 ROS基础



The screenshot shows the CLion 1.0.2 IDE interface. The main editor displays a C++ file named `main.cpp` with the following code:

```
#include <iostream>

using namespace std;

int main() {
    int i=20;
    cin>>i;
    cin>>i;
    cin>>i;

    return 0;
}
```

The left sidebar shows the project structure for `Test1` (F:\Workspace\Clion\Test1), including `CMakeLists.txt`, `main.cpp`, `Tree.h`, and `External Libraries`.

The bottom panel shows the Run output for `Test1`. The command executed is `C:\Users\zxs\.clion10\system\cmake\generated\f28f7873\f28f7873\Debug\Test1.exe`. The output shows three lines of input (1, 1, 2) and three lines of output (2, 2, 3). The process finished with exit code 0.

