上海科技大学
ShanghaiTech University

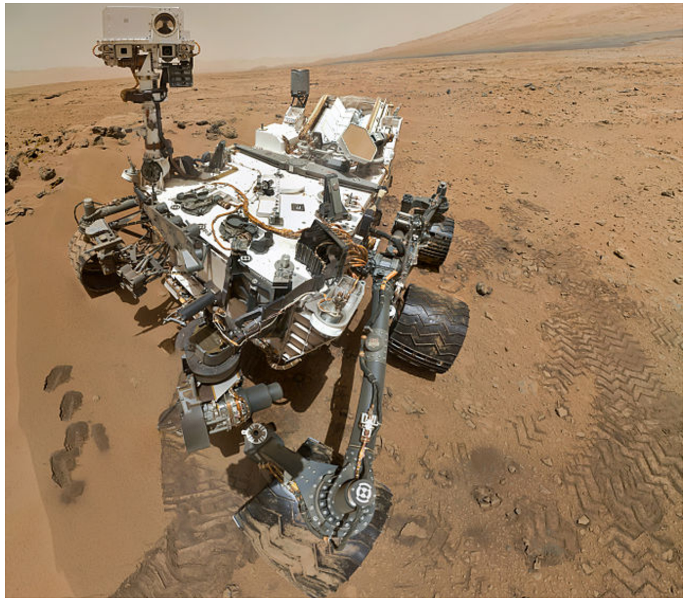
CS283: **Robotics Fall 2016: Software**

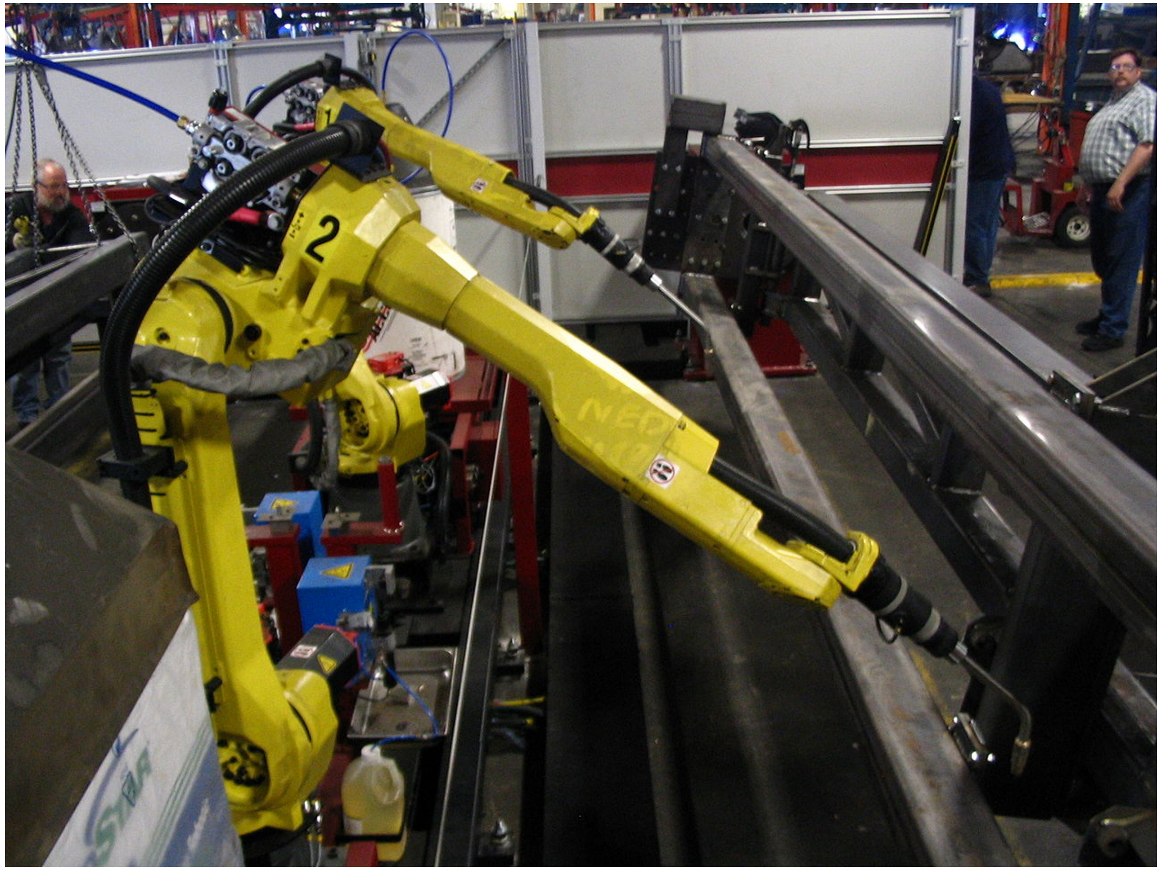
Sören Schwertfeger / 师泽仁

ShanghaiTech University

Review

- Definition Robot: A machine capable of performing complex tasks in the physical world, that is using sensors to perceive the environment and acts tele-operated or autonomous.
- Usually Industrial Robots are stationary.
- Most other Robots move.





Most important capability

(for autonomous mobile robots)

How to get from A to B?

(safely and efficiently)

- Autonomous mobile robots move around in the environment. Therefore **ALL** of them:
  - They need to know **where** they **are**.
  - They need to know **where** their **goal** is.
  - They need to know **how** to get there.

- Where am I?
  - GPS, Guiding system
  - Build a map: Mapping
  - Find position in a map: Localization
  - Both: Simultaneous Localization and Mapping (SLAM)
- Where is my goal?
  - What is the goal: map or object recognition
  - Where is that goal?

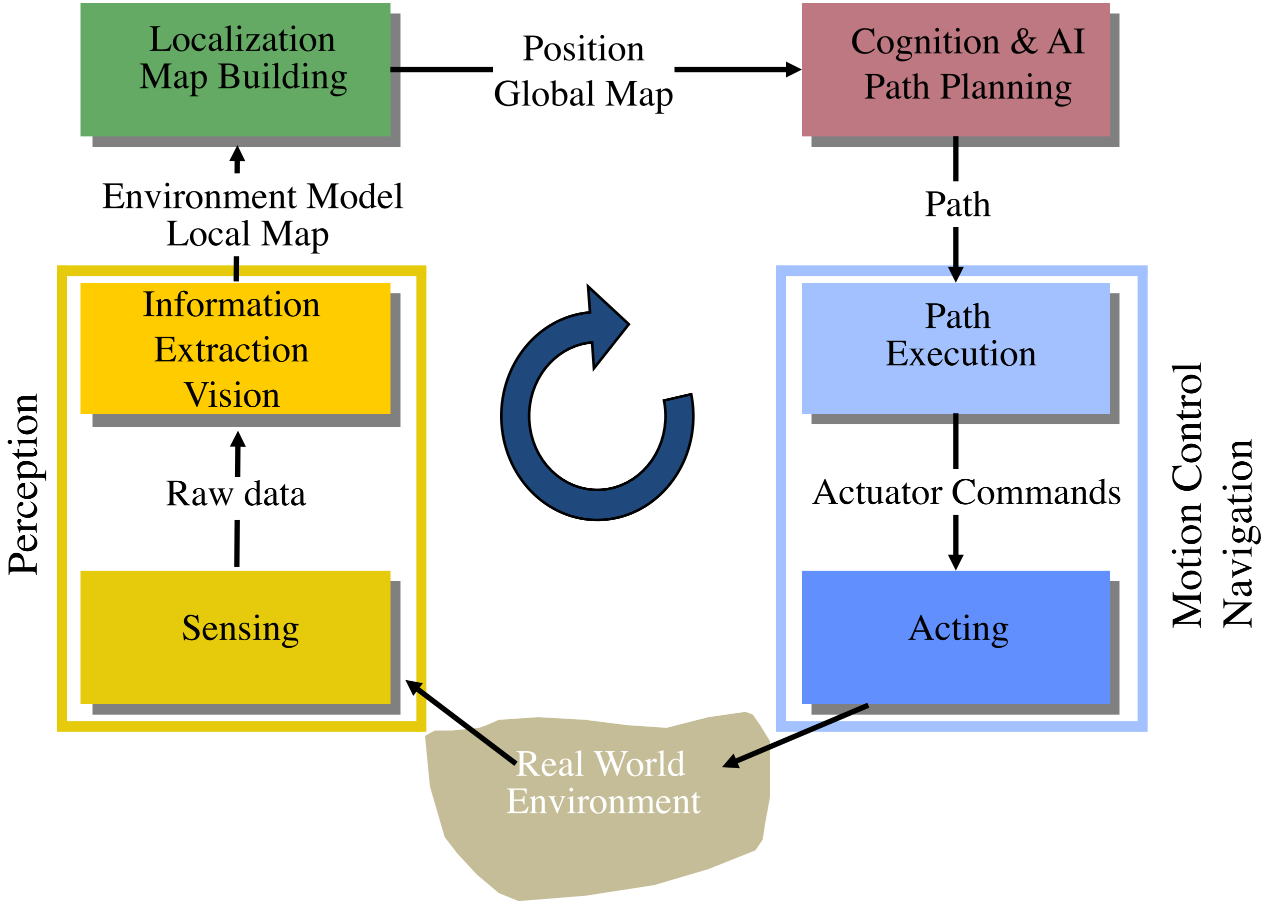
- Autonomous mobile robots move around in the environment. Therefore **ALL** of them:
  - They need to know **where** they **are**.
  - They need to know **where** their **goal** is.
  - They need to know **how** to get there.

- Different levels:
  - Control:
    - How much power to the motors to move in that direction, reach desired speed
  - Navigation:
    - Avoid obstacles
    - Classify the terrain in front of you
    - Predict the behavior (motion) of other agents (humans, robots, animals, machines)
  - Planning:
    - Long distance path planning
    - What is the way, optimize for certain parameters

How to get from A to B?

How to program an intelligent ROBOT to go from A to B?

General Control Scheme for Mobile Robot Systems

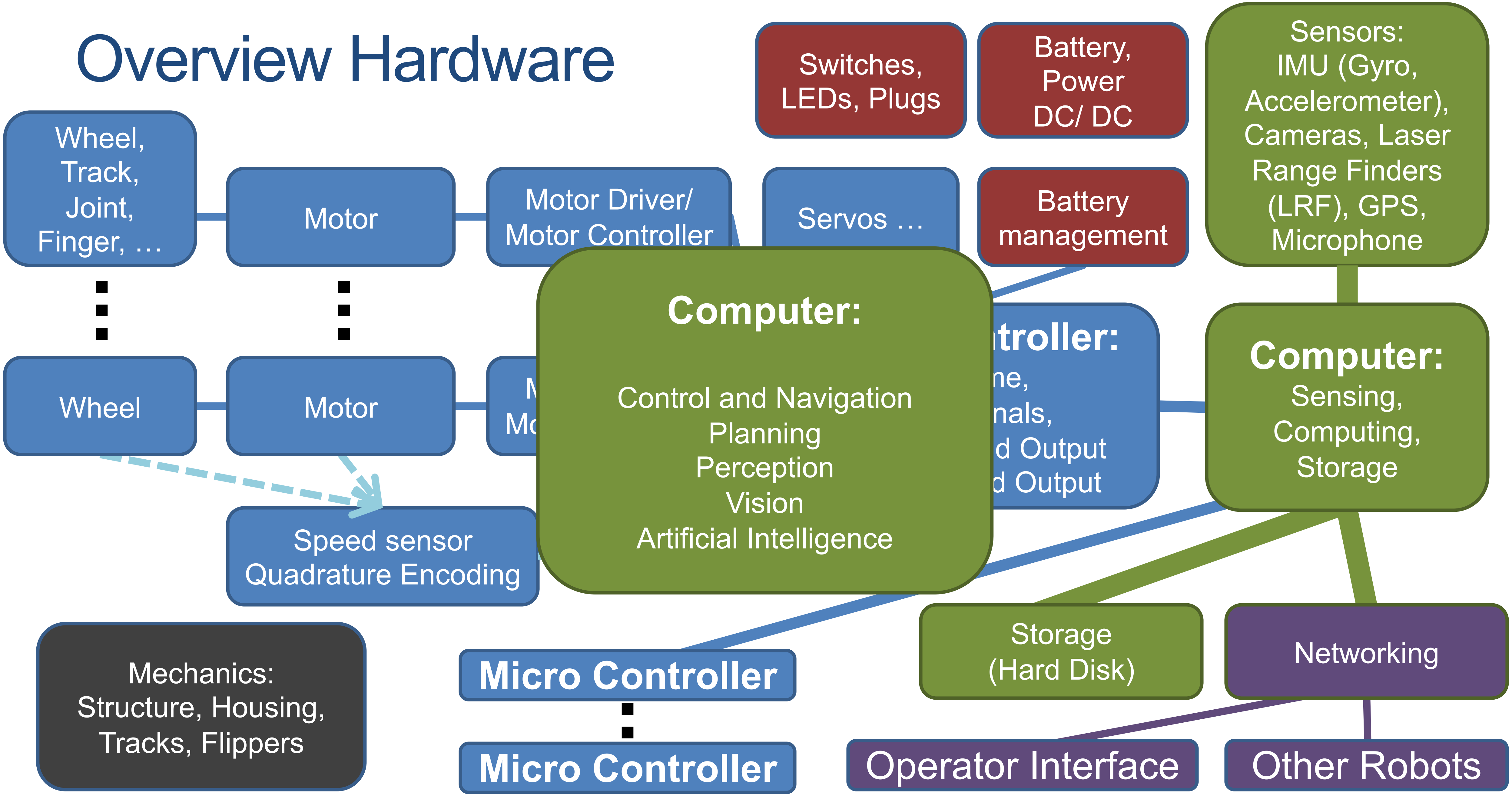


With material from Roland Siegwart and Davide Scaramuzza, ETH Zurich

How to get from A to B?

What are the components of a ROBOT?

Overview Hardware



Outline

- Software
  - Software Design
  - Programming Review
  - Robot Operating System (ROS)

Robot Software: Tasks/ Modules/ Programs (ROS: node)

Support

- Communication with Micro controller
- Sensor drivers
- Networking
  - With other PCs, other Robots, Operators
- Data storage
  - Store all data for offline processing and simulation and testing
- Monitoring/ Watchdog

Robotics

- Control
- Navigation
- Planning
- Sensor data processing
  - e.g. Stereo processing, Image rectification
- Mapping
- Localization
- Object Recognition
- Mission Execution
- Task specific computing, e.g.:
  - View planning, Victim search, Planning for robot arm, …

Software Design

- Modularization:
  - Keep different software components separated
  - ☺ Keep complexity low
  - ☺ Easily exchange a component (with a different, better algorithm)
  - ☺ Easily exchange multiple components with simulation
  - ☺ Easily exchange dada from components with replay from hard disk instead of live sensor data
  - ☺ Multiple programming teams working on different components easier
  - Need: Clean definition of interfaces or exchange messages!
  - Allows: Multi-Process (vs. Single-Process, Multi-Thread) robot software system
  - Allows: Distributing computation over multiple computers

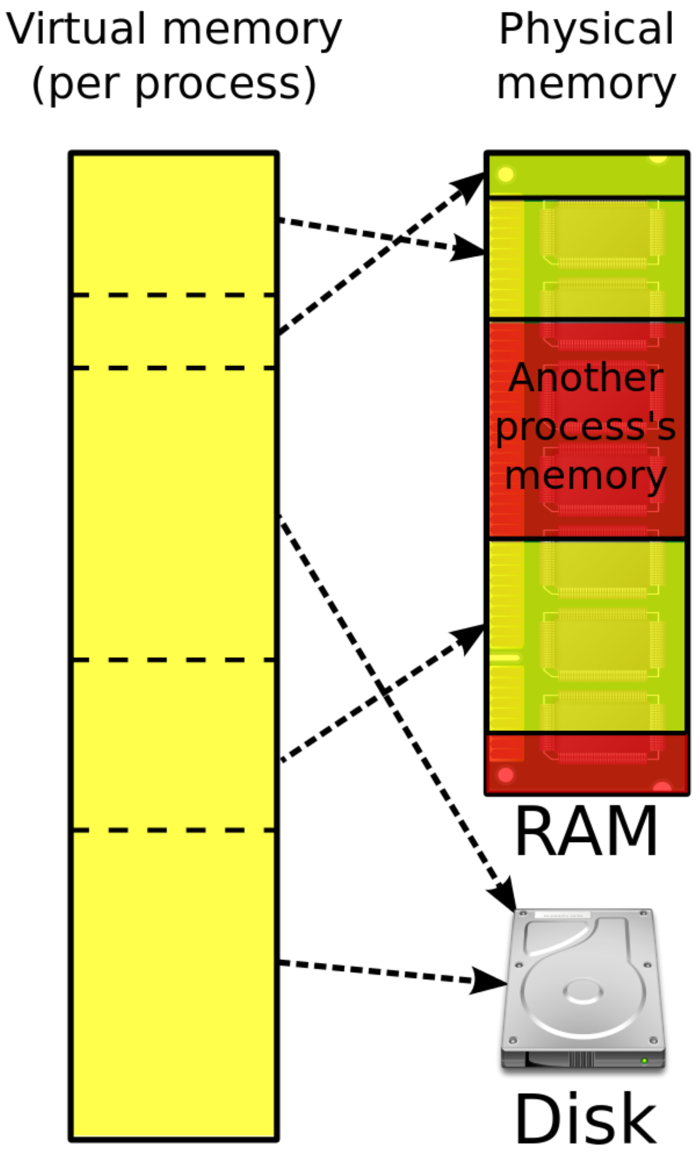
Programming review

- Process vs. Thread
- C++ Object Orientation
- Constant Variables
  - const-correctness
- C++ Templates
- Shared Pointer

- Objective:
  - Prerequisites for understanding ROS.
  - Understand how we can efficiently retrieve and transfer data in ROS.

Process

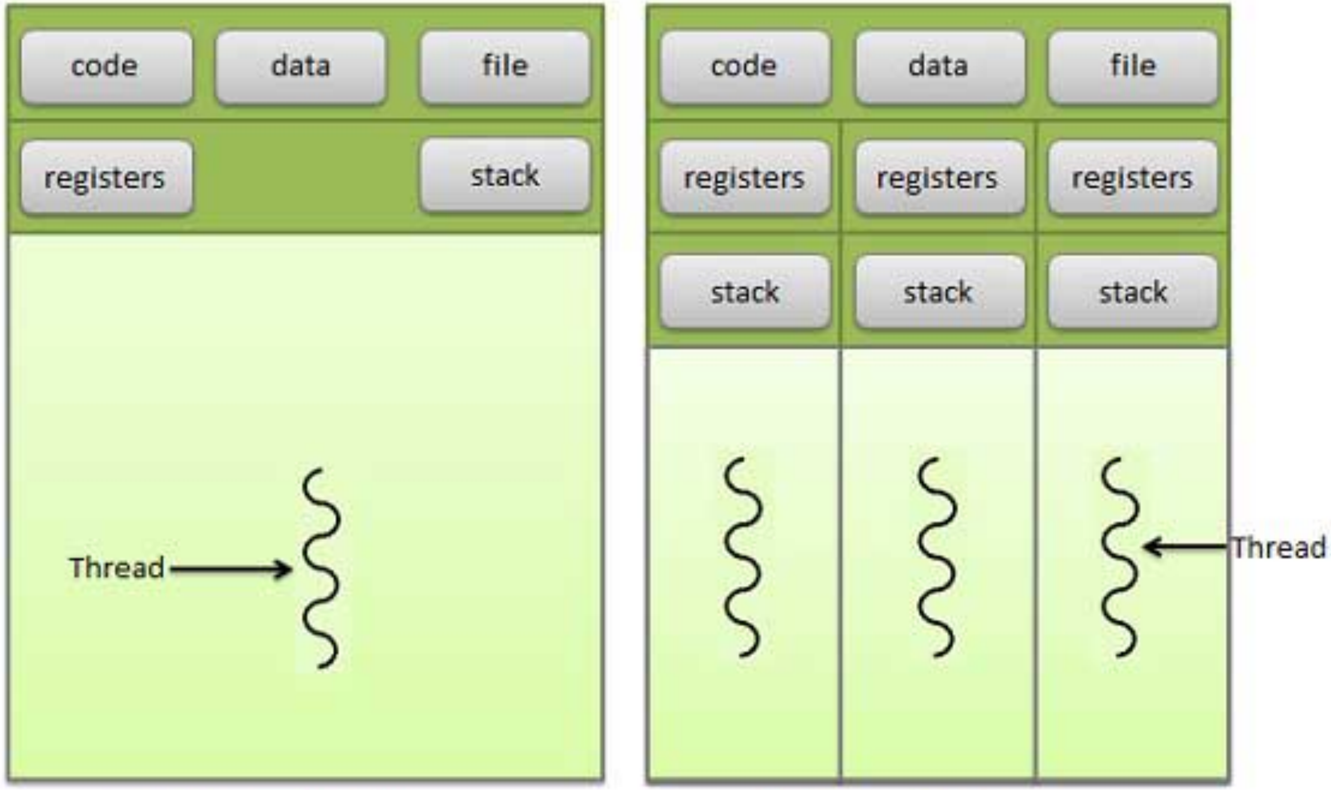
- Execution of one instance of a computer program
- Virtual memory:
  - Contains only code and data from this program, the libraries and the operating system
  - Other processes (programs) can not access this memory (shared memory access is possible but complicated)
- Operating system gives each process equal amount of processing time (scheduling) – if the processes need it
  - Good support from the operating system to give certain processes higher or lower priority
  - Linux console program to see processes: **top**



(From Wikipedia)

Multi-Threading

- In one process, multiple threads => parallel execution
- ☺ Code and Memory is shared => easy exchange of data, save mem.
- 😐 Synchronization can be tricky (mutex, dead lock, race condition)
- ☹ If one thread crashes, the whole process (all threads) die



Single threaded Process Multi-threaded Process

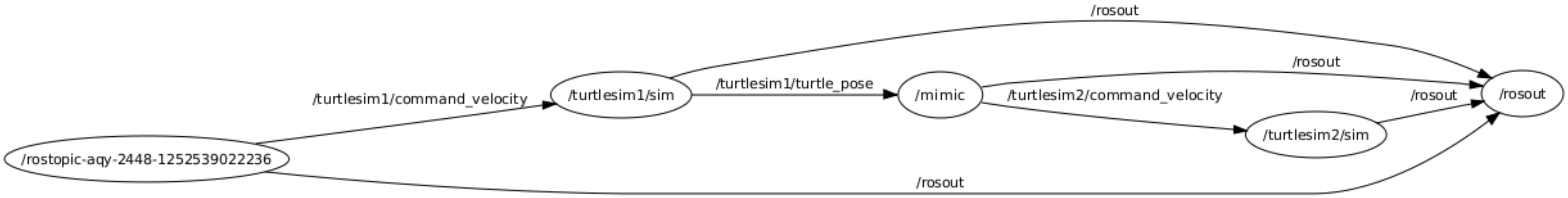
(from http://www.tutorialspoint.com)

Processes and Threads in Robotics - Messages

- Both approaches have been implemented!
- Both are used and important!
- Robot Operating System (ROS): Multiple Processes:
  - Each component runs in its own process: called **node**
  - A node can have multiple threads => faster computation
  - Nodes communicate using **messages**
  - A node can send (**publish**) **messages** under different names called **topic**
  - Nodes can listen to (**subscribe**) **messages** under different **topics**
  - The messages are transferred over the network (TCP/IP) => multiple computers work together transparently
  - ☹ Messages are serialized, copied and de-serialized even if both nodes on the same computer => slow (compared to pointer passing)
    - Optimization: **Nodelet**: run different nodes in the SAME process => pointer passing => fast

ROS nodes

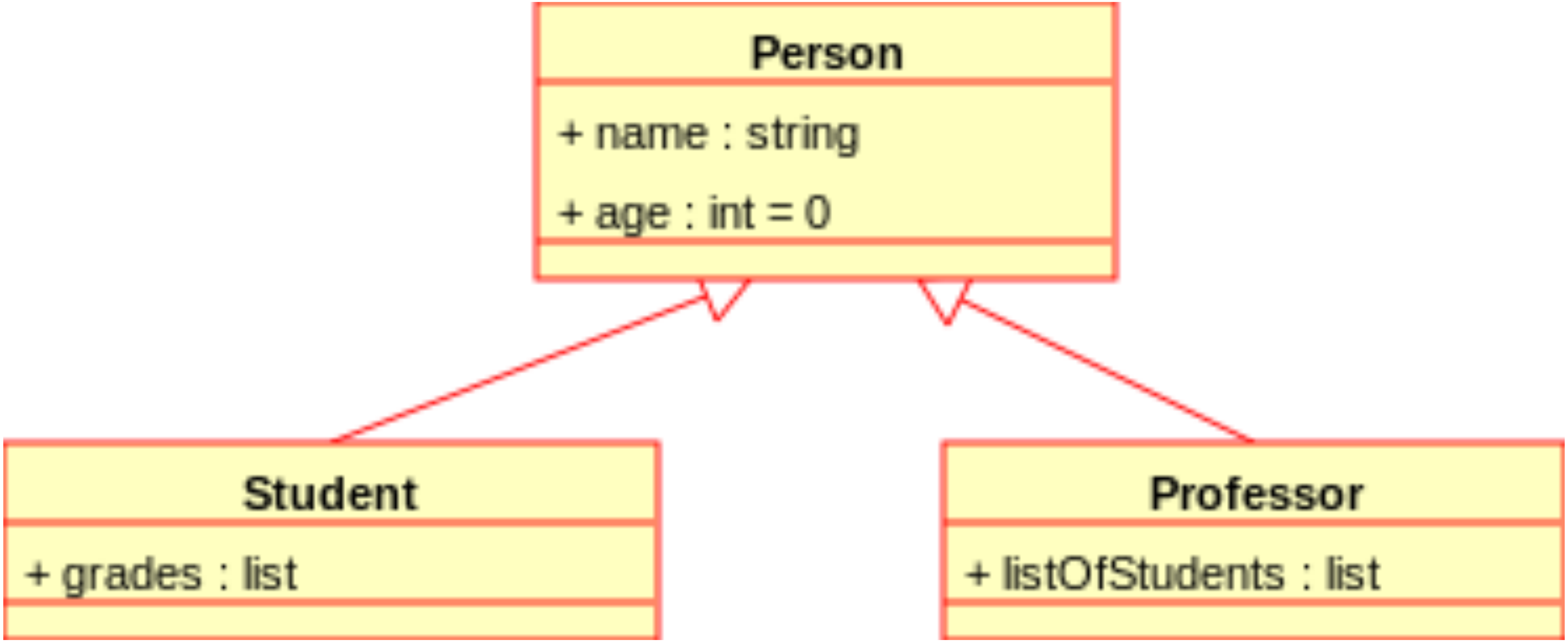
- **ROS core**: keep track which **nodes** are running and their **topics**
- Show all nodes and topics in a graph: `rosrun rqt_graph rqt_graph`
  - `/rosout` : special node for output on console (standard out)
  - `/turtlesim1/sim, /turtlesim2/sim` : simulated robots (**nodes**) (multiple nodes per simulated robot)
  - `/command_velocity` : set the speed of a robot (**topic**)
  - **Node** `/turtlesim1/sim` **publishes** on **topic** `/turtlesim1/turtle_pose`
  - **Node** `/mimic` **subscribes** to **topic** `/turtlesim1/turtle_pose`



Object Oriented (OO) Programming

- C++ is OO … C is not
- Object: have data fields (variables) and associated procedures (methods)
- Instance of an object: created with keyword **new**
- Object: Abstract data type: has data and code
  - encapsulation and information hiding: private variables not visible for outside code – interact through the methods
  - Methods can be private, too: can only be used by (methods of) the object itself
  - Inheritance: code-reuse through re-use of variables and methods from base class. Child class extends/ modifies functionality
  - Polymorphism: Base class defines interface to some functionality (e.g. Method for getting a camera image). A child implements the actual code for a specific use case (e.g. A certain driver for a specific camera) – this is **NOT** how ROS works
    - ROS uses **messages** as “interface”
- Objects have destructors for deletion/ cleanup

Object Orientation: Example



(From Wikipedia)

Constant Variables

- Declare variables that do not change (anymore) in the code: `const`
- Works for variables and objects
- Const Objects:
  - Only methods that do not change any variable of the object may be called =>
  - Those methods have to be declared const
- Used for program-correctness
- Especially for multi-threading:
  - Share the data (e.g. image)
  - Make it read only via const
  - => no side-effects between different threads

1. const int x = 5; // x may not be changed
2. int * someValue = &x; // pointer – compilation error!!
3. const int * pointy = &x; // good
4. *pointy = 8; // error – pointing to const!
5. int y = 4;
6. pointy = &y; // from non const to const is always possible!
7. const int * p2 const = &y; // pointing to const variable and p2 is also const
8. p2 =&x; // error – p2 is const

QUESTIONS REGARDING HW1?

Admin

- Did you read your Literature?
  - Will be provided at least one week ahead from now on.

- Please join piazza
  - Please use your ping yin name

- HW 1:
  - Don’t forget to send me your public ssh key on Thursday already!
  - Backup both private and public ssh keys!

C++ Templates

- Functions and classes that operate with generic types
- Function or class works on many different data types without rewrite
  - `template <typename T> int compare( T v1, T v2);`
  - Type of T is determined during compile time => errors during compilation (and not run-time)
  - Any type (type == class) that offers the needed methods & variables can be used
  - Usage: `compare<string>( string("string number one"), "hello world" );`
    - Explicit declaration: typename T = string
    - typename T can (most often) deducted by the compiler from the argument types
- Class template:

```
template <typename T> class myStuff{
    T v1, v2;
    myStuff(T var1, T var2){  v1 = var2; v2 = var2; }
};
```

Template example

```
//This example throws the following error : call of overloaded 'max(double, double)' is ambiguous
template <typename Type>
Type max(Type a, Type b) {
    return a > b ? a : b;
}
```

```
#include <iostream>

int main(int, char**)
{
  // This will call max <int> (by argument deduction)
  std::cout << max(3, 7) << std::endl;
  // This will call max<double> (by argument deduction)
  std::cout << max(3.0, 7.0) << std::endl;
  // This type is ambiguous, so explicitly instantiate max<double>
  std::cout << max<double>(3, 7.0) << std::endl;
  return 0;
}
```

Shared Pointer

- C++ Standard Library (std): heavily templated part of C++ Standard (many parts used to be in boost library)
- Pointer: address of some data in the heap – in the virtual address space
- Space for data has to be allocated (reserved) with: `new`
- After usage of data it has to be destroyed to free the memory: `delete`
- Problem: Data (e.g.) image is shared among different modules/ components/ threads. Who is the last user – who has to delete the data?
  - Shared pointer: counts the number of users (smart pointers); upon destruction of last user (smart pointer) the object gets destroyed : called "Reference counting"
  - Problem: Shared pointer needs to know the destructor method for the pointer =>
  - Shared pointer is a templated class: Template argument: class type of the object pointed to
  - Shared pointer can also point to const object!

Shared pointer example

```
std::shared_ptr<int> p1(new int(5));
std::shared_ptr<int> p2 = p1; //Both now own the memory.

p1.reset(); //Memory still exists, due to p2.
p2.reset(); //Deletes the memory, since no one else owns the memory.
```

- Earlier, shared_ptr used to be in boost
- Excerpt from ROS message of type “String” :

```
typedef boost::shared_ptr< ::std_msgs::String_<ContainerAllocator> > Ptr;
typedef boost::shared_ptr< ::std_msgs::String_<ContainerAllocator> const> ConstPtr;
```

- typedef: create another (shorter) name for a certain type
- Our type: a shared pointer that points to a (complicated) String object

```
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}
```

Review for ROS

- Different components, modules, algorithms run in different processes: **nodes**
- Nodes communicate using **messages** (and **services** …)
- Nodes **publish** and **subscribe** to **messages** by using names (**topics**)
- **Messages** are often passed around as shared pointers which are
  - “write protected” using the const keyword
  - The shared pointers take the message type as template argument
  - Shared pointers can be accessed like normal pointers

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

int main(int argc, char **argv){
  ros::init(argc, argv, "talker");
  ros::NodeHandle n;

  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

  ros::Rate loop_rate(10);
  int count = 0;
  while (ros::ok()){
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    chatter_pub.publish(msg);

    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

ROS Tutorial: Listener

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

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);

  ros::spin();

  return 0;
}
```