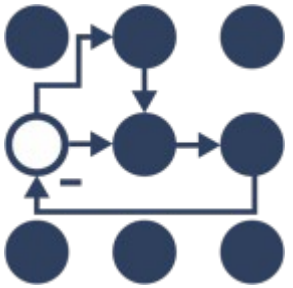
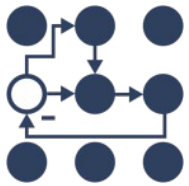


ROS control, an overview

ROSCon 2014
September 12, 2014
Adolfo Rodríguez Tsouroukdissian

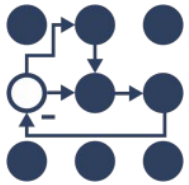
PAL ROBOTICS



A show of hands

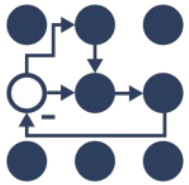
have you ever...

- **used** a controller / robot driver **not** written by you?
- **implemented** a controller / robot driver **yourself**?
  - subject to **realtime** constraints?



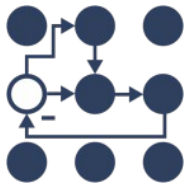
ROS control – an overview

- Big picture and goals
- ROS control & friends
  - Setting up a robot
  - Controllers
  - The control loop
- Demo
- Robots using ROS control

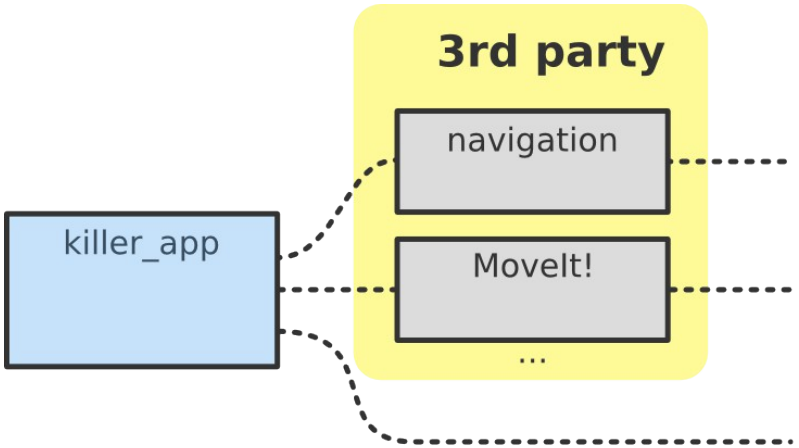


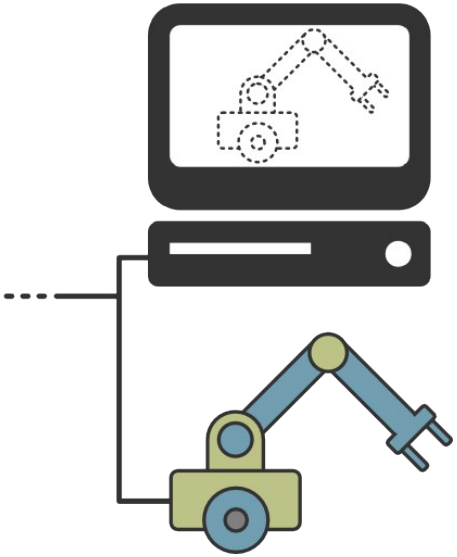
ROS control – an overview

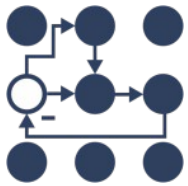
- Big picture and goals
- ROS control & friends
  - Setting up a robot
  - Controllers
  - The control loop
- Demo
- Robots using ROS control



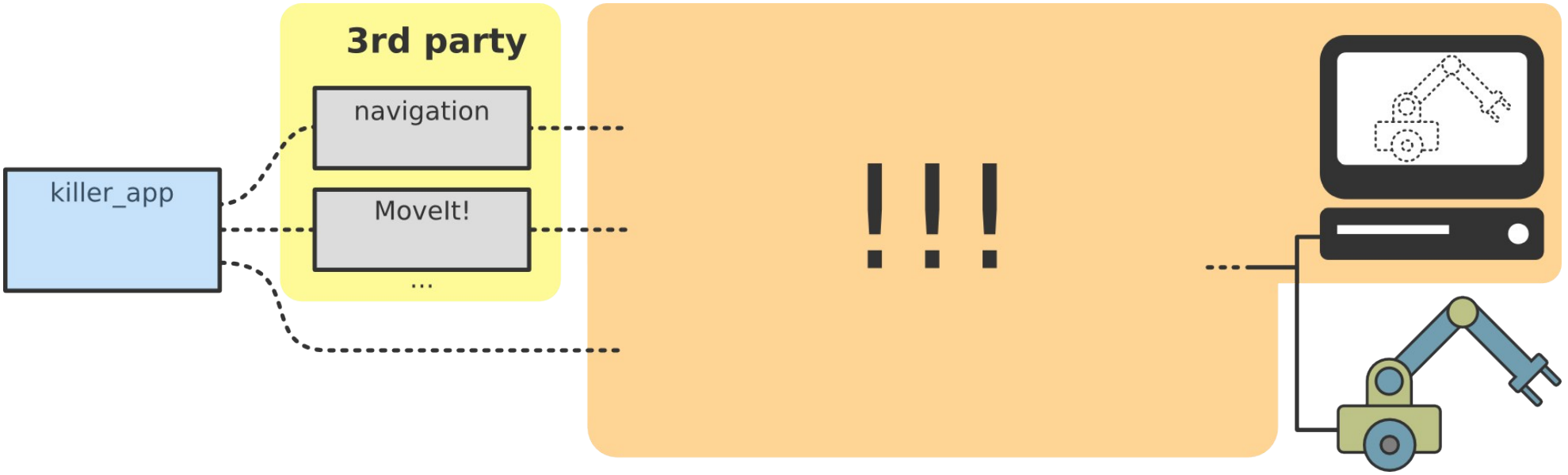
ROS control – the big picture

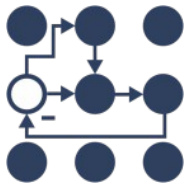




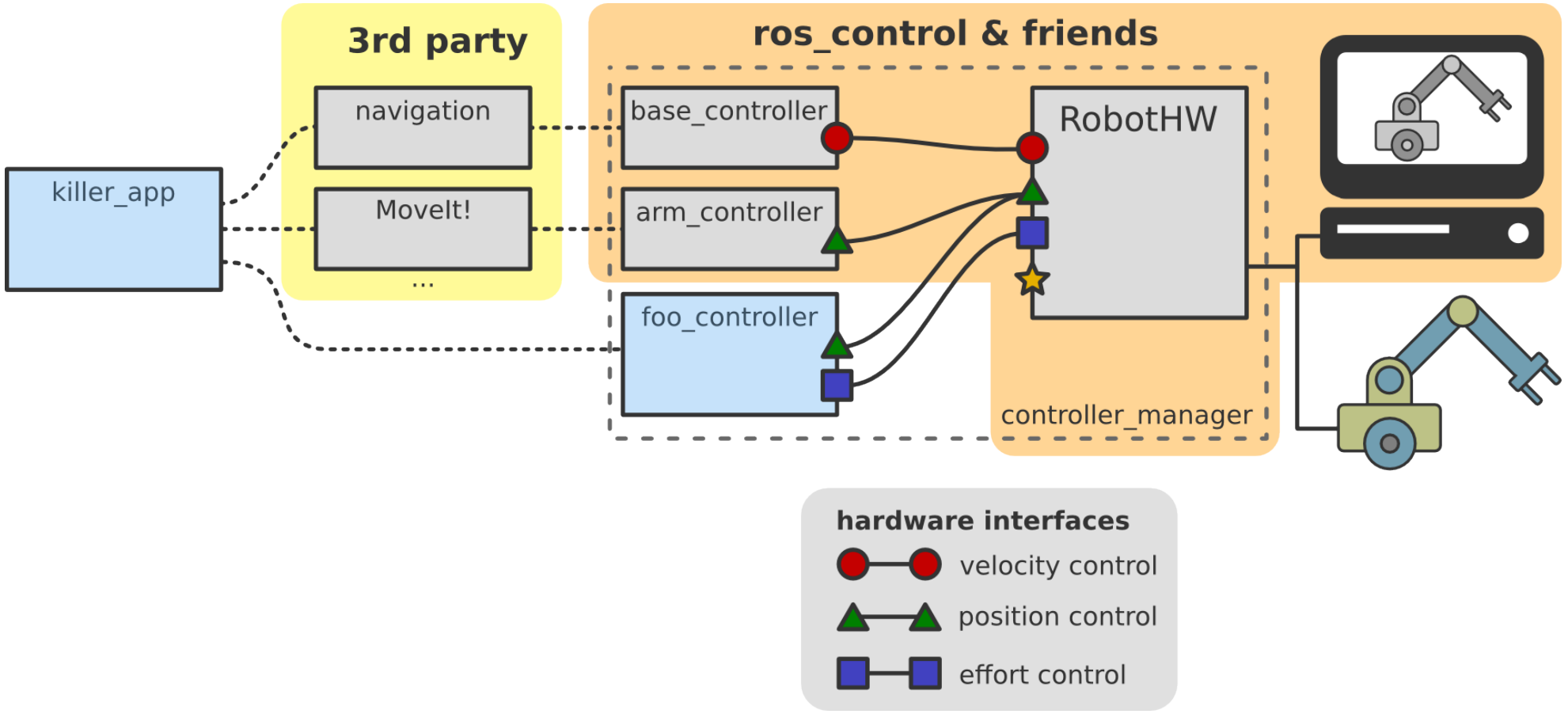


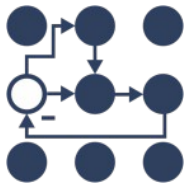
ROS control – the big picture



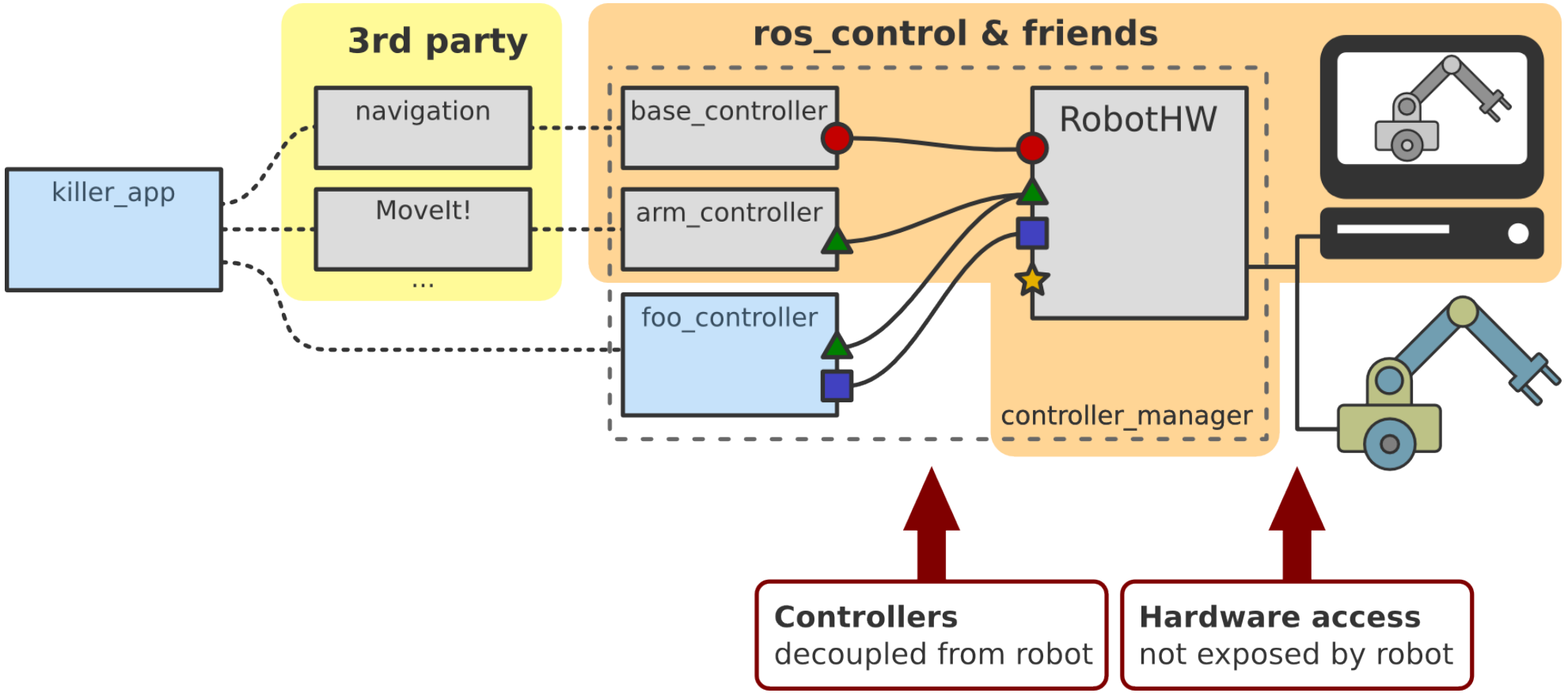


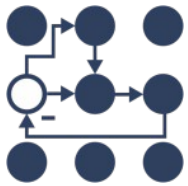
ROS control – the big picture



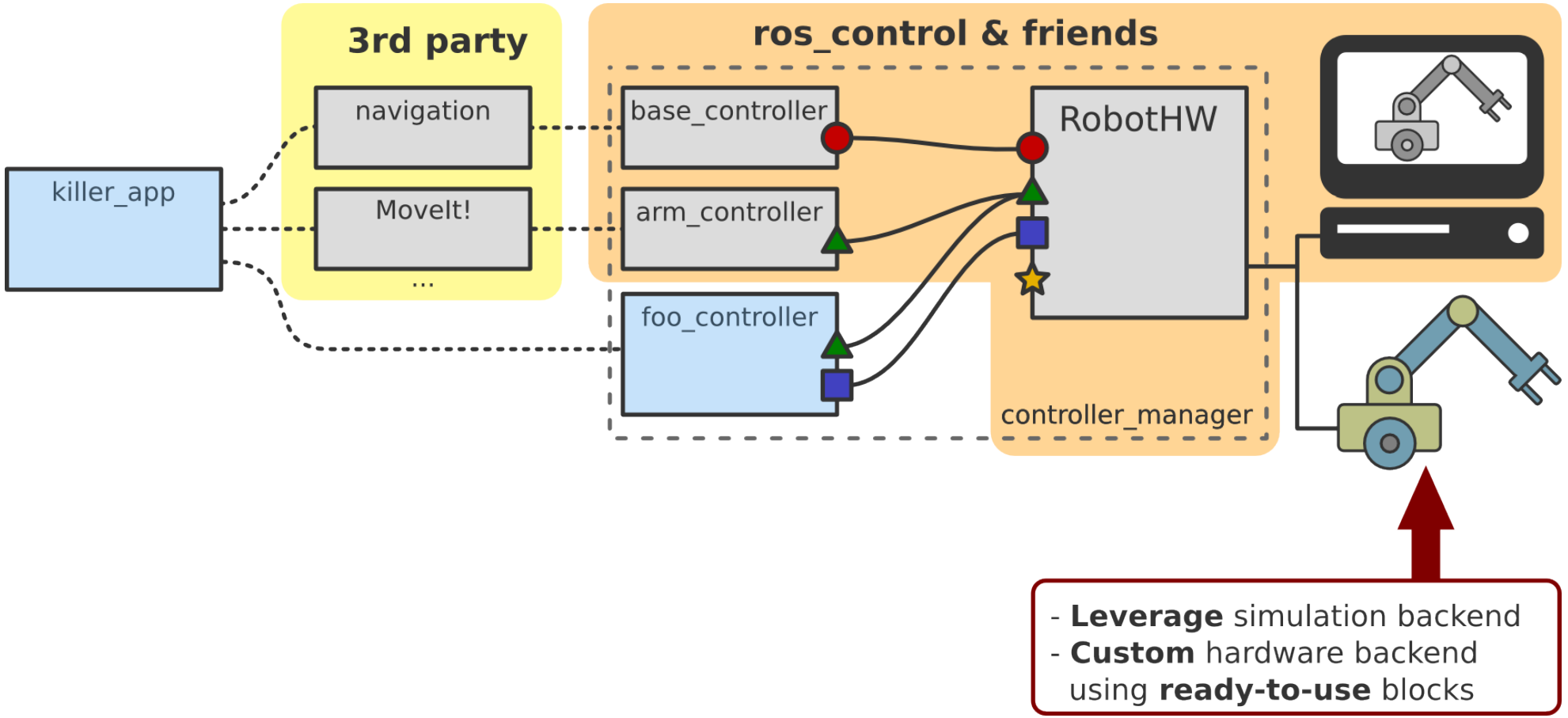


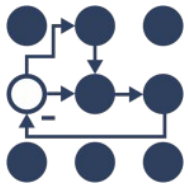
ROS control – the big picture



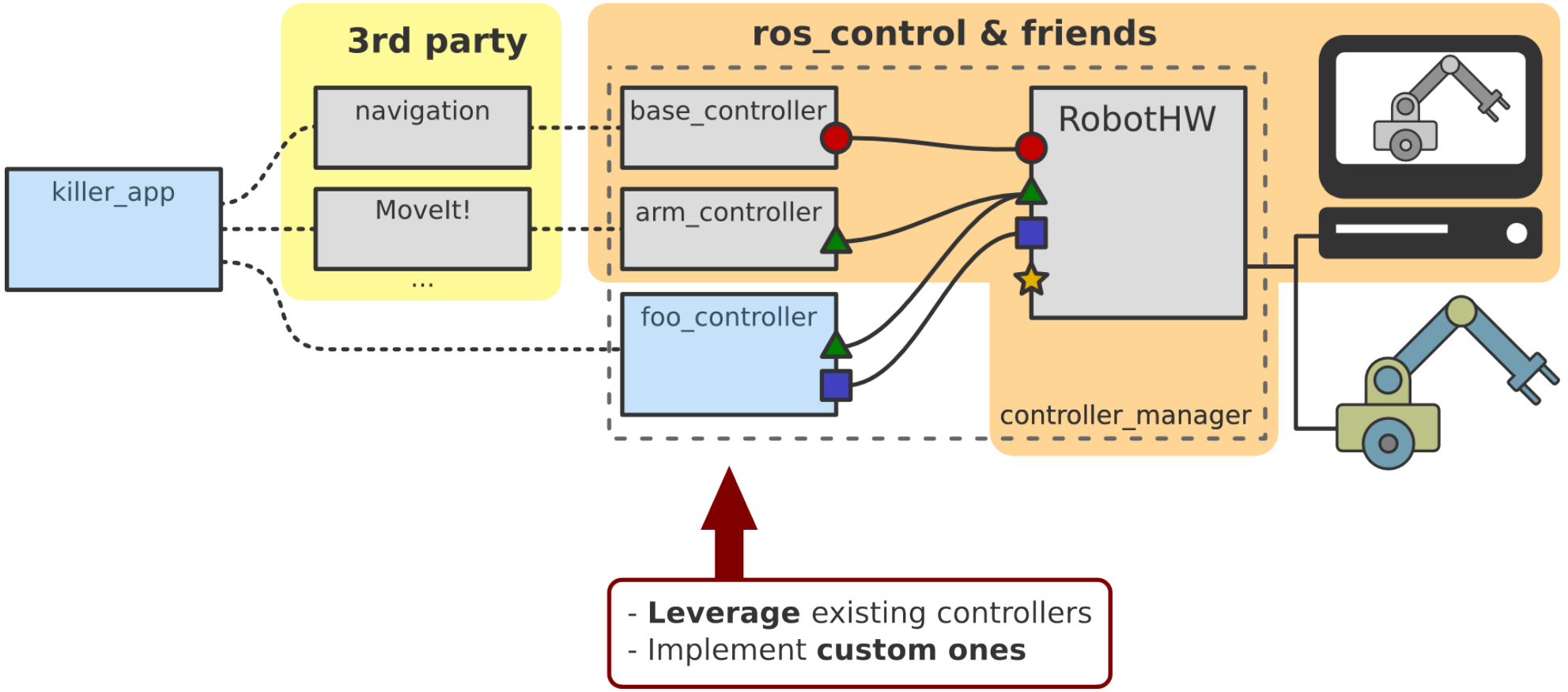


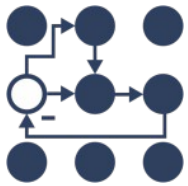
ROS control – the big picture



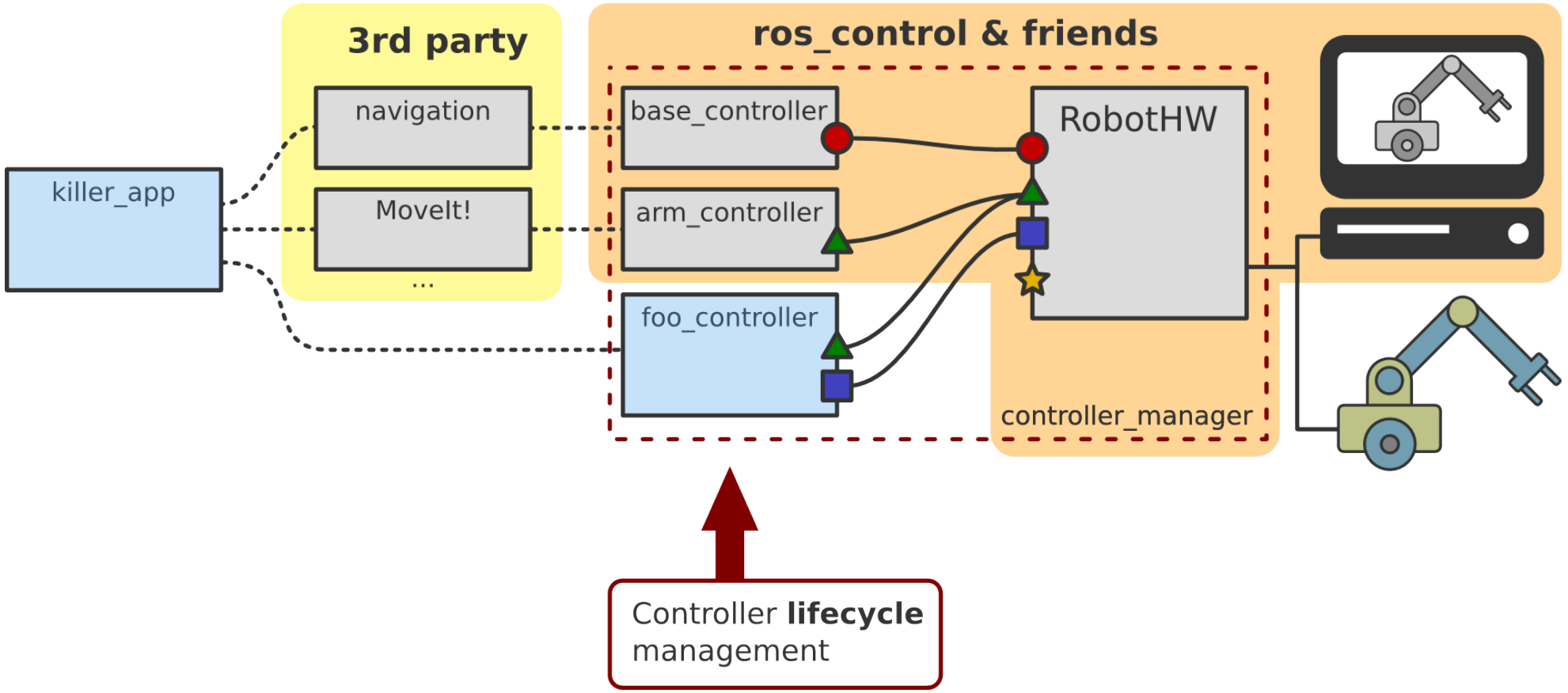


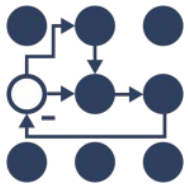
ROS control – the big picture



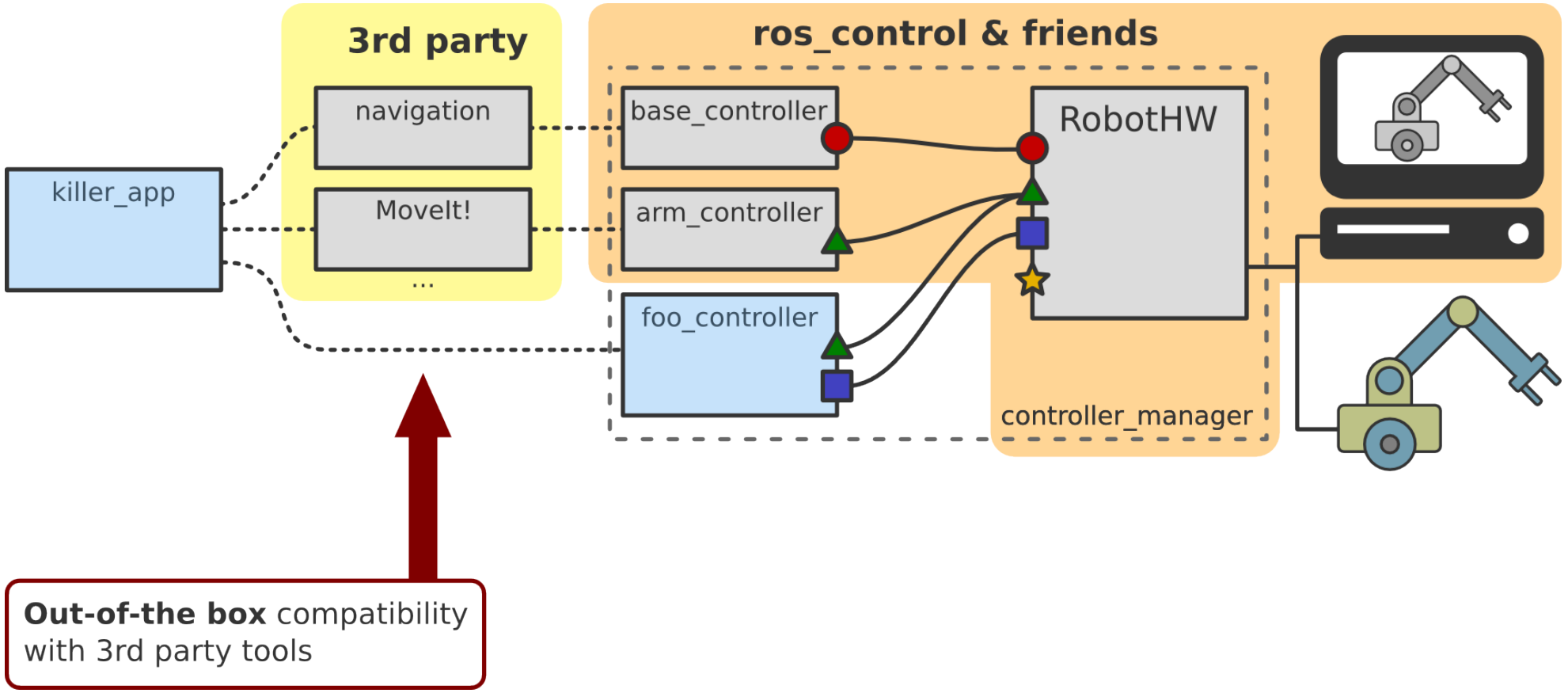


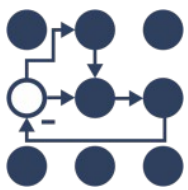
ROS control – the big picture



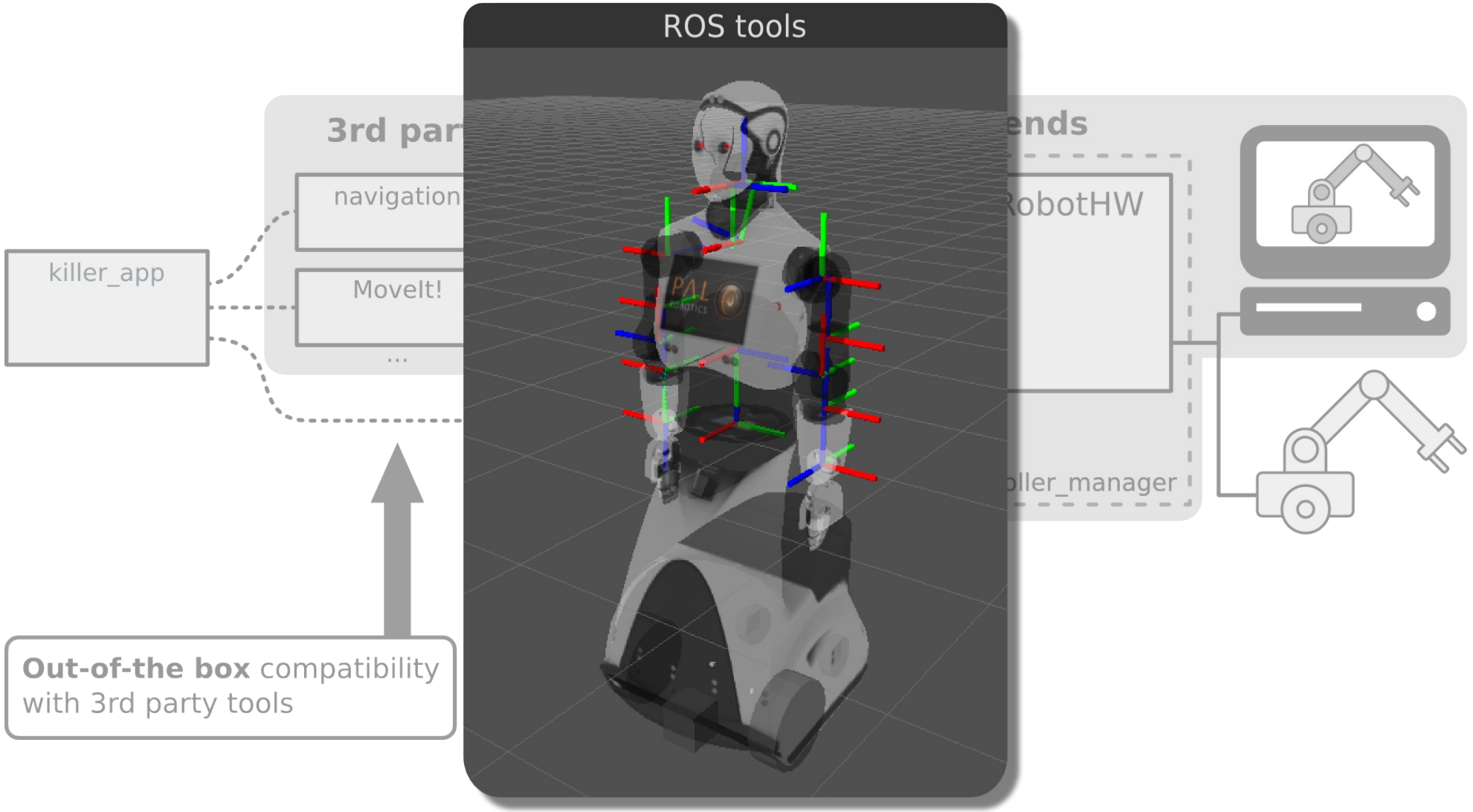


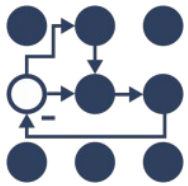
ROS control – the big picture



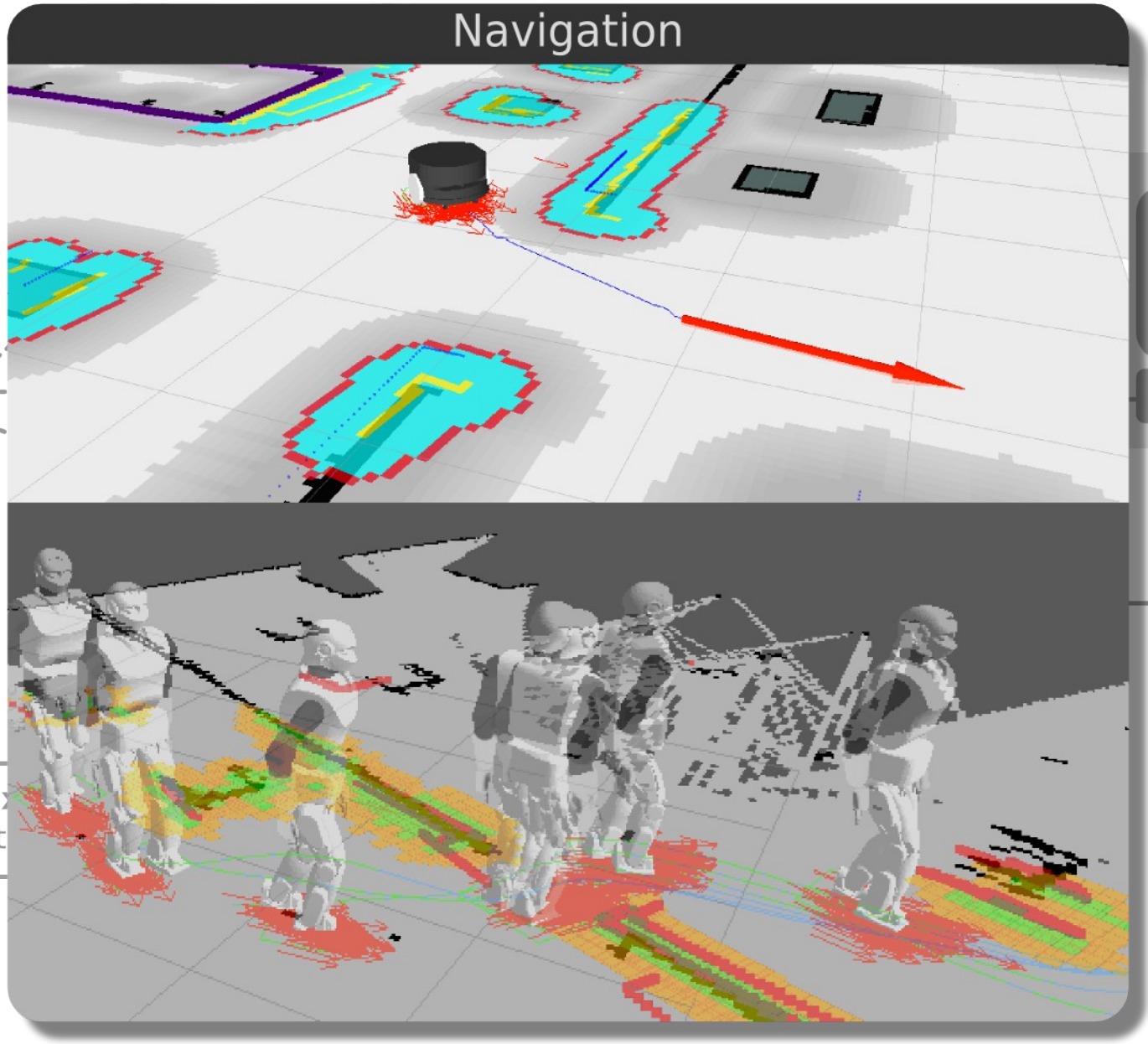


ROS control – the big picture



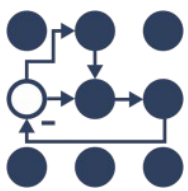


ROS control – the big picture

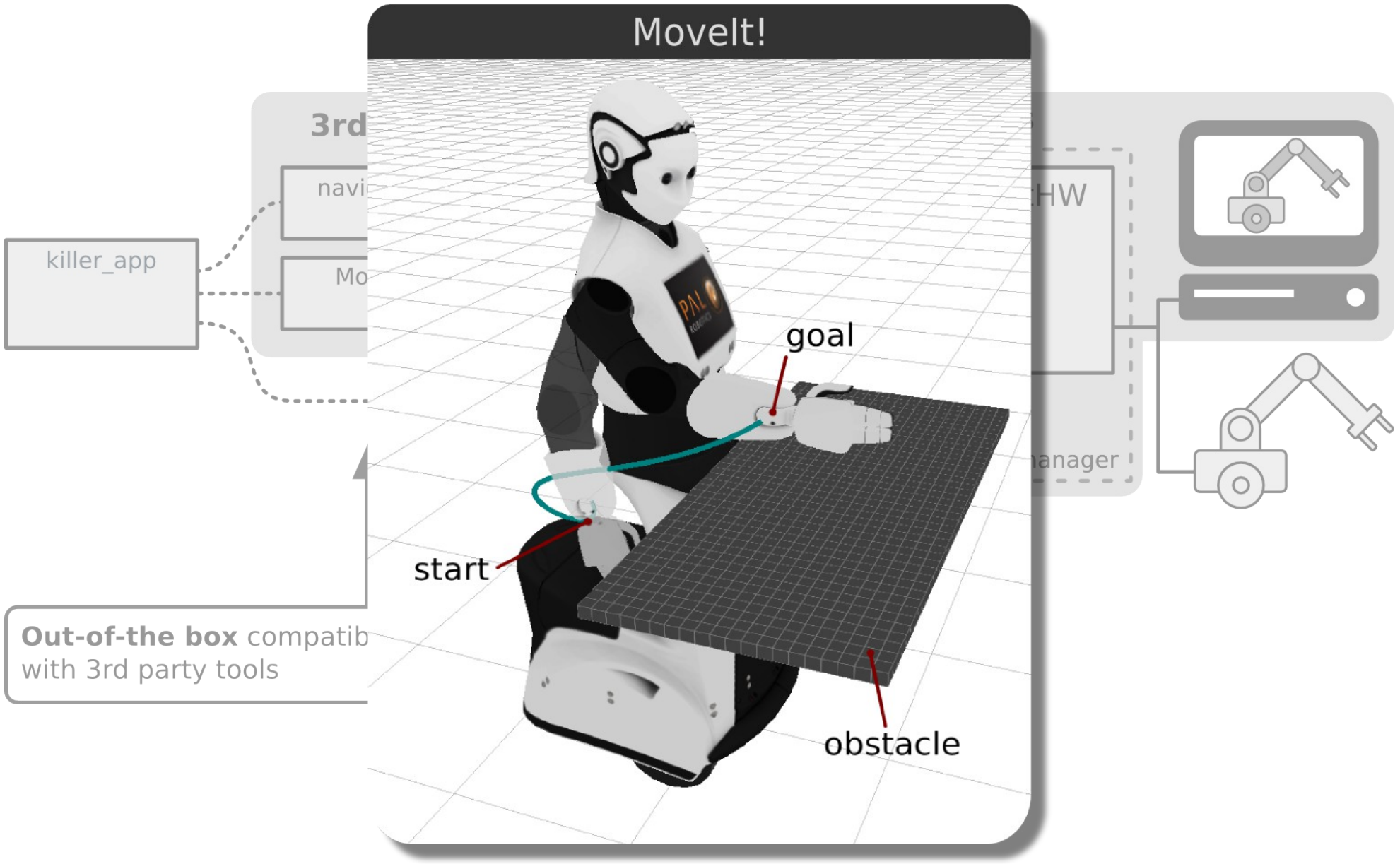


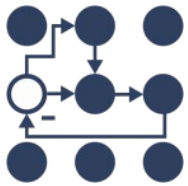
killer_app

Out-of-the box
with 3rd party t

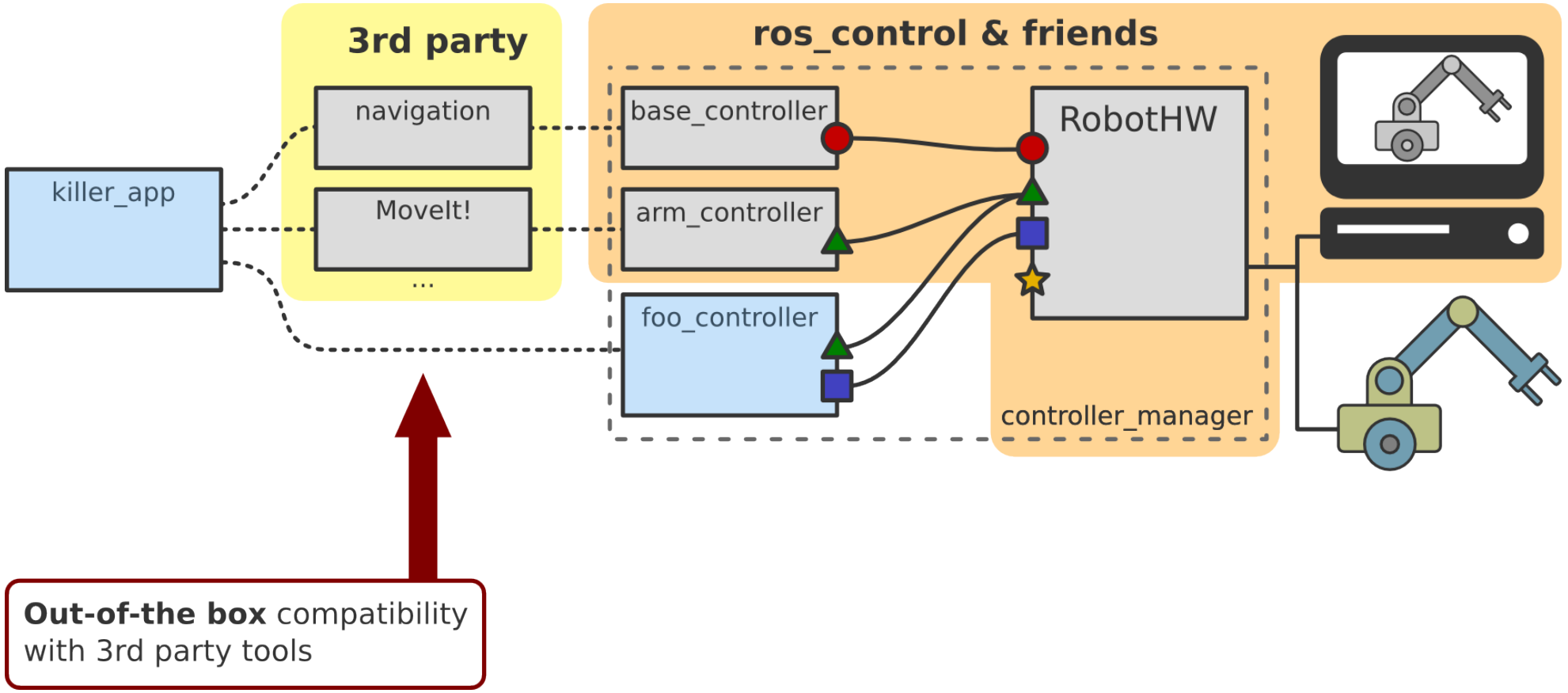


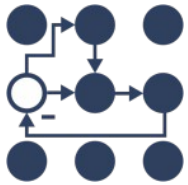
ROS control – the big picture



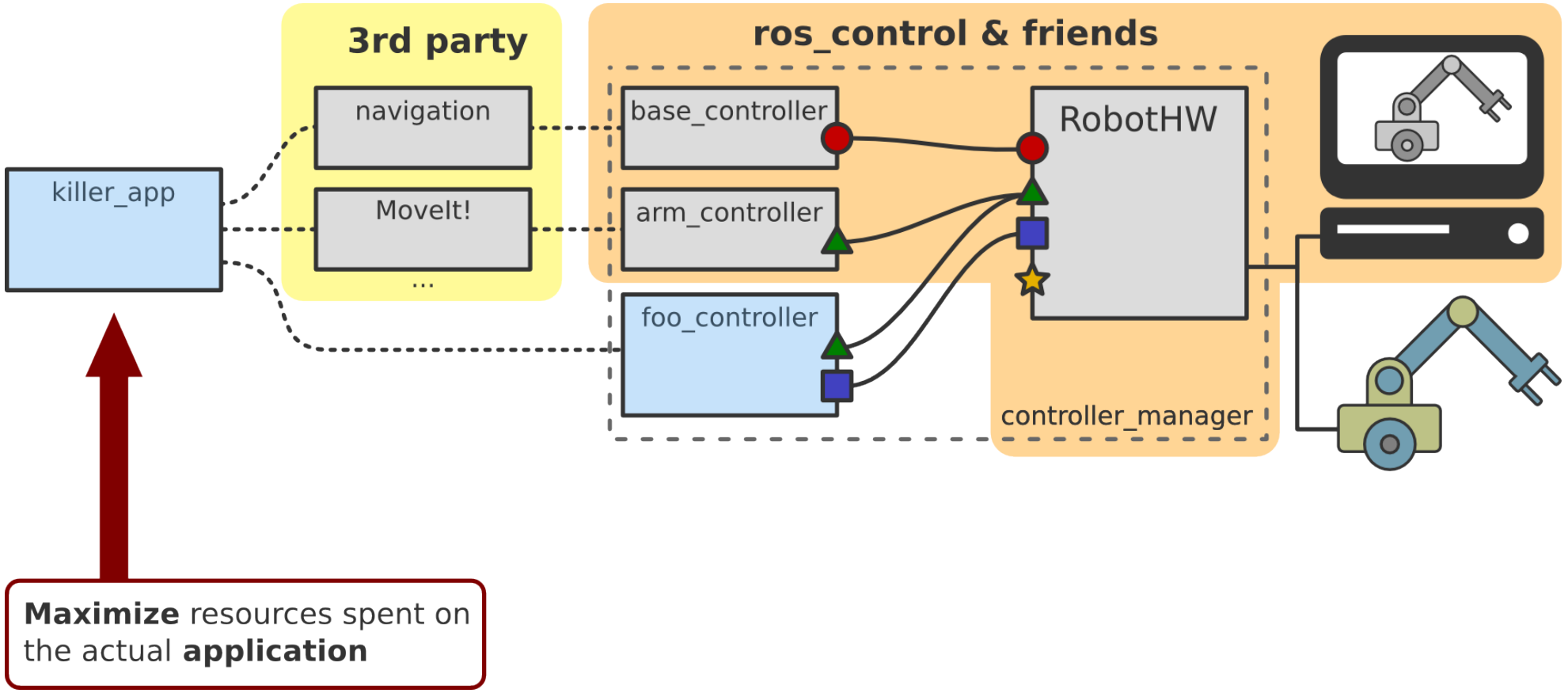


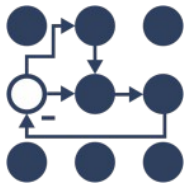
ROS control – the big picture



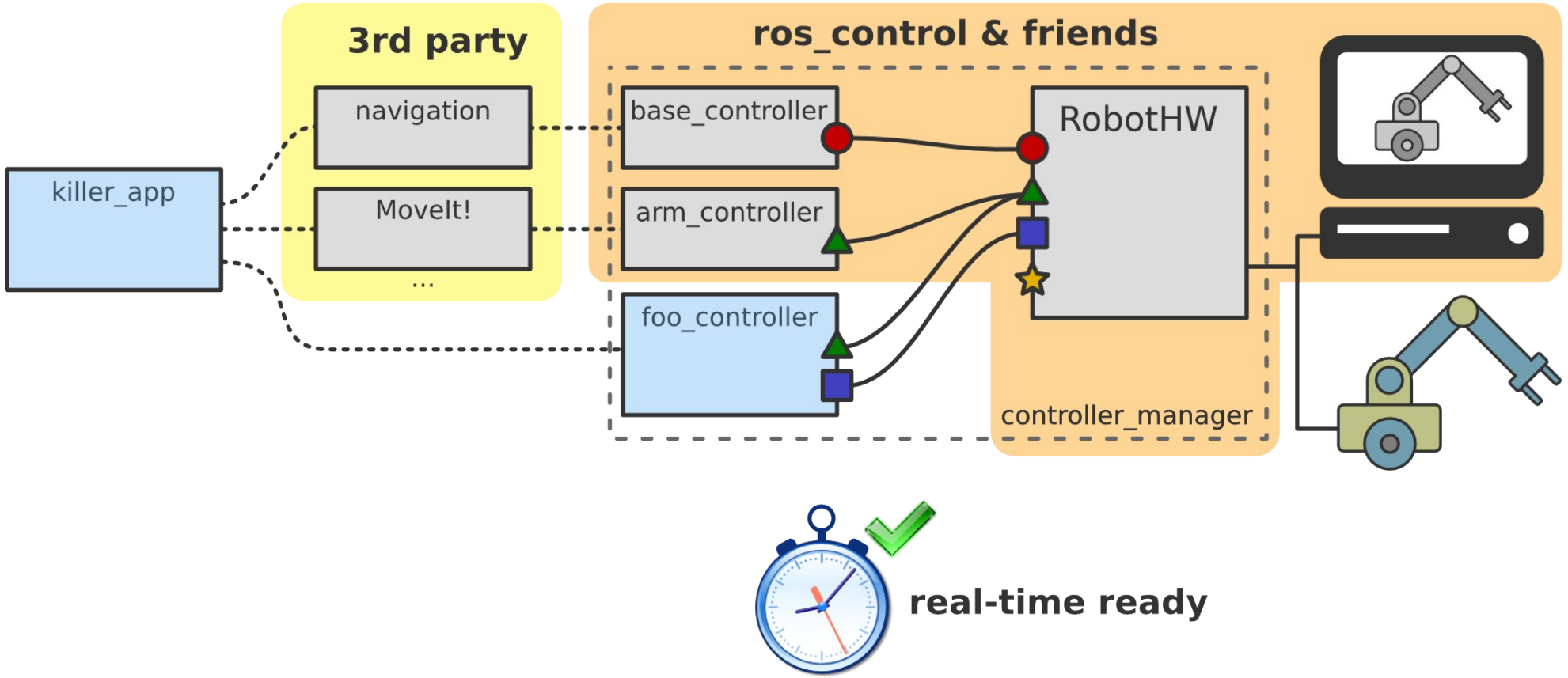


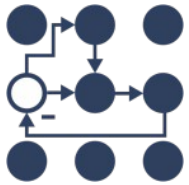
ROS control – the big picture





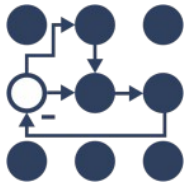
ROS control – the big picture





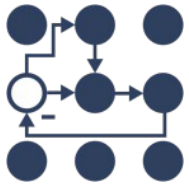
ROS control – goals

- Lower **entry barrier** for exposing HW to ROS
- Promote **reuse** of control code
- Provide **ready-to-use** tools
- **Real-time ready** implementation



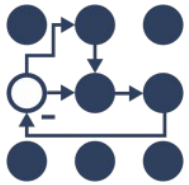
History

- **pr2_controller_manager** (2009)
  - developed mainly by **Willow Garage** (WG)
  - **PR2**-specific
- **ros_control** (late 2012)
  - started by **hiDOF**, in collaboration with **WG**
  - continued by **PAL Robotics** and **community**
  - **robot-agnostic** version of the pr2_controller_manager



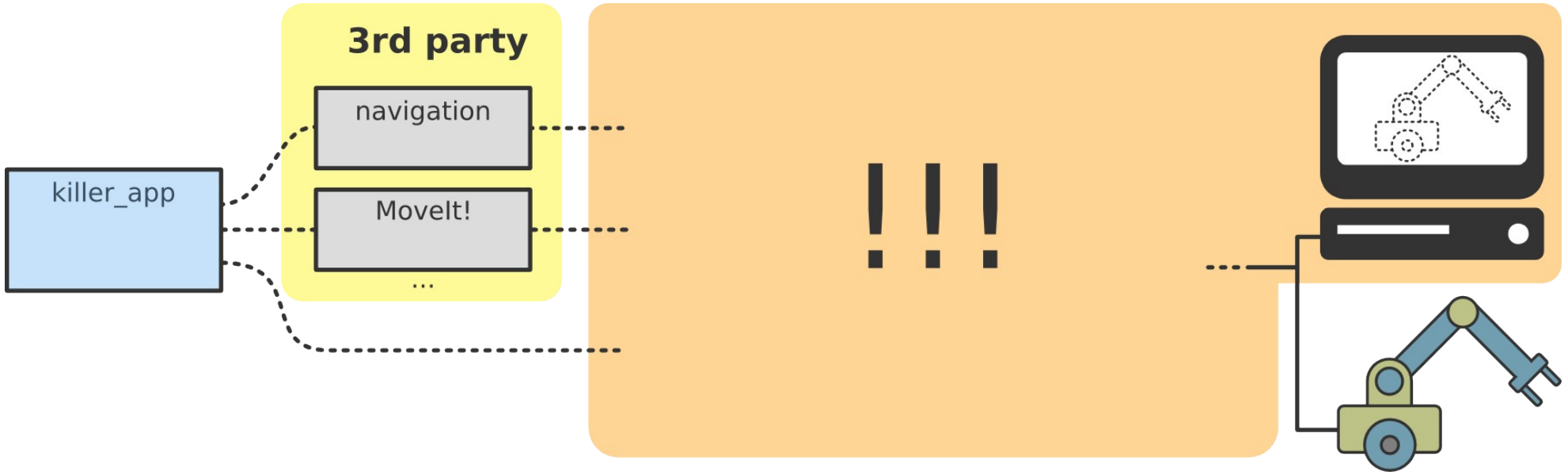
Related work

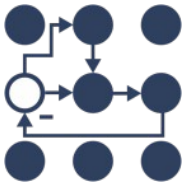
- Component-based frameworks
  - Orocos RTT
    - ROCK
    - Conman (exposes controller_manager ROS API)
  - OpenRTM
  - YARP (real-time WIP)
- Computational graph models
  - Ecto
  - Microblx
  - many, many, non-robotics implementations



Related work

- No **end-to-end** solution for ROS users (AFAIK)
  - bring standard ROS interfaces **closer to hardware**
  - move integration effort to the **driver level**





code repositories

ros-controls

github.com/ros-controls

control_msgs

messages and actions useful for controlling robots

realtime_tools

tools that can be used from a hard realtime thread

control_toolbox

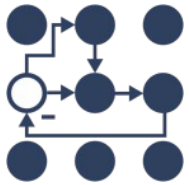
tools useful for writing controllers and robot abstractions

ros_control

generic and basic controller framework for ROS

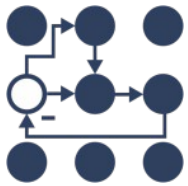
ros_controllers

generic robot controllers for ros_control

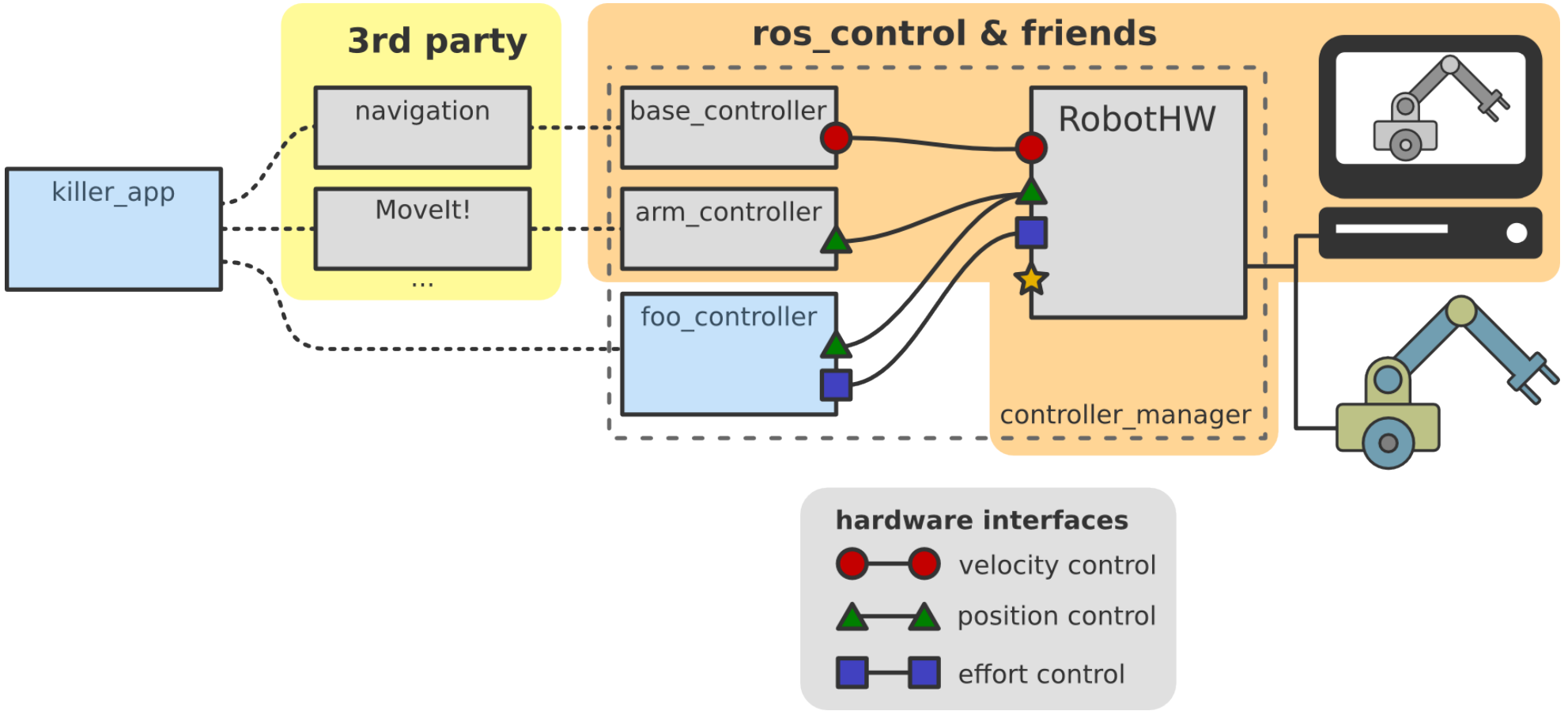


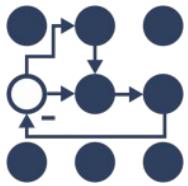
ROS control – an overview

- Big picture and goals
- ROS control & friends
  - **Setting up a robot**
  - Controllers
  - The control loop
- Demo
- Robots using ROS control

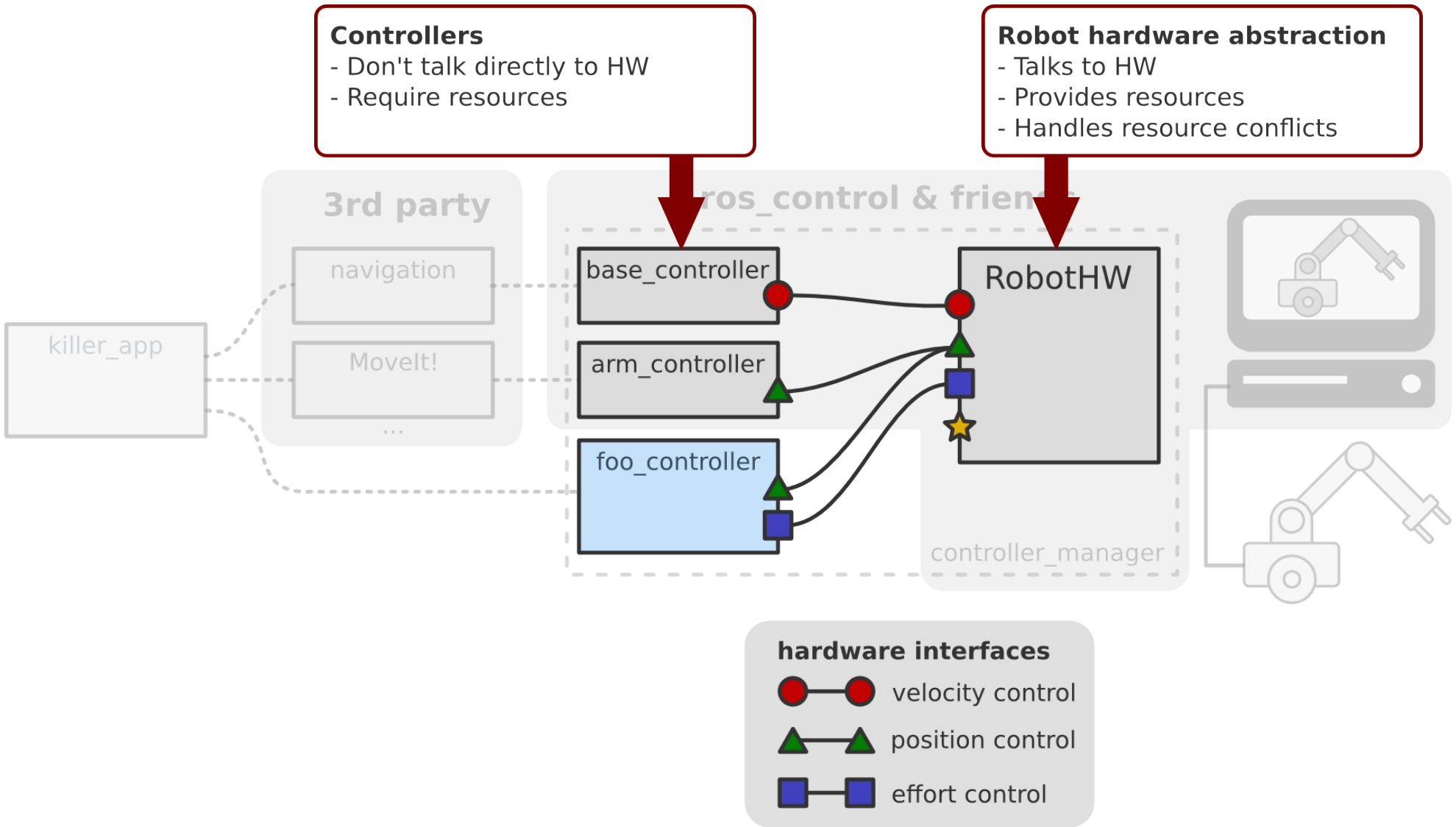


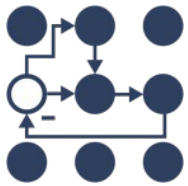
Setting up a robot



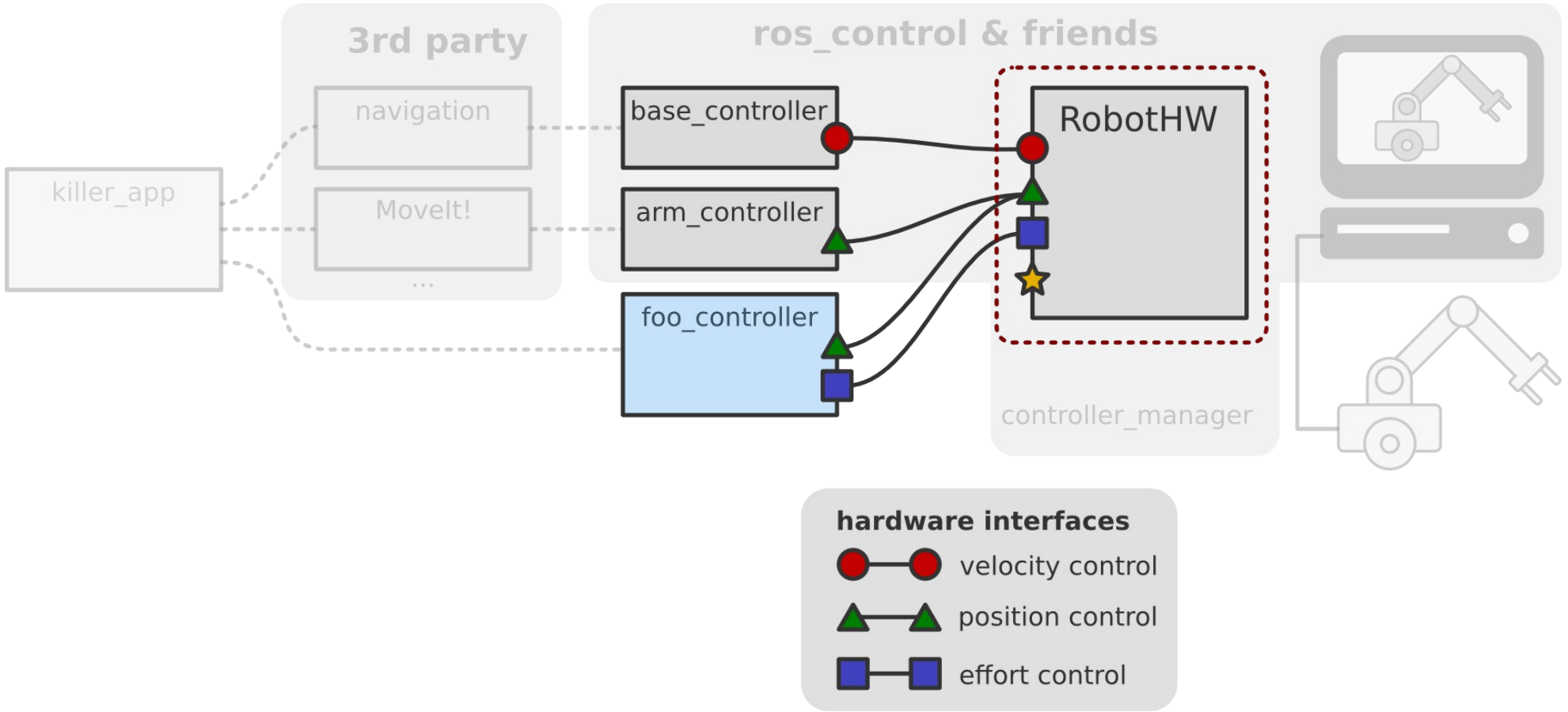


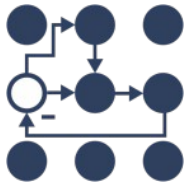
Setting up a robot



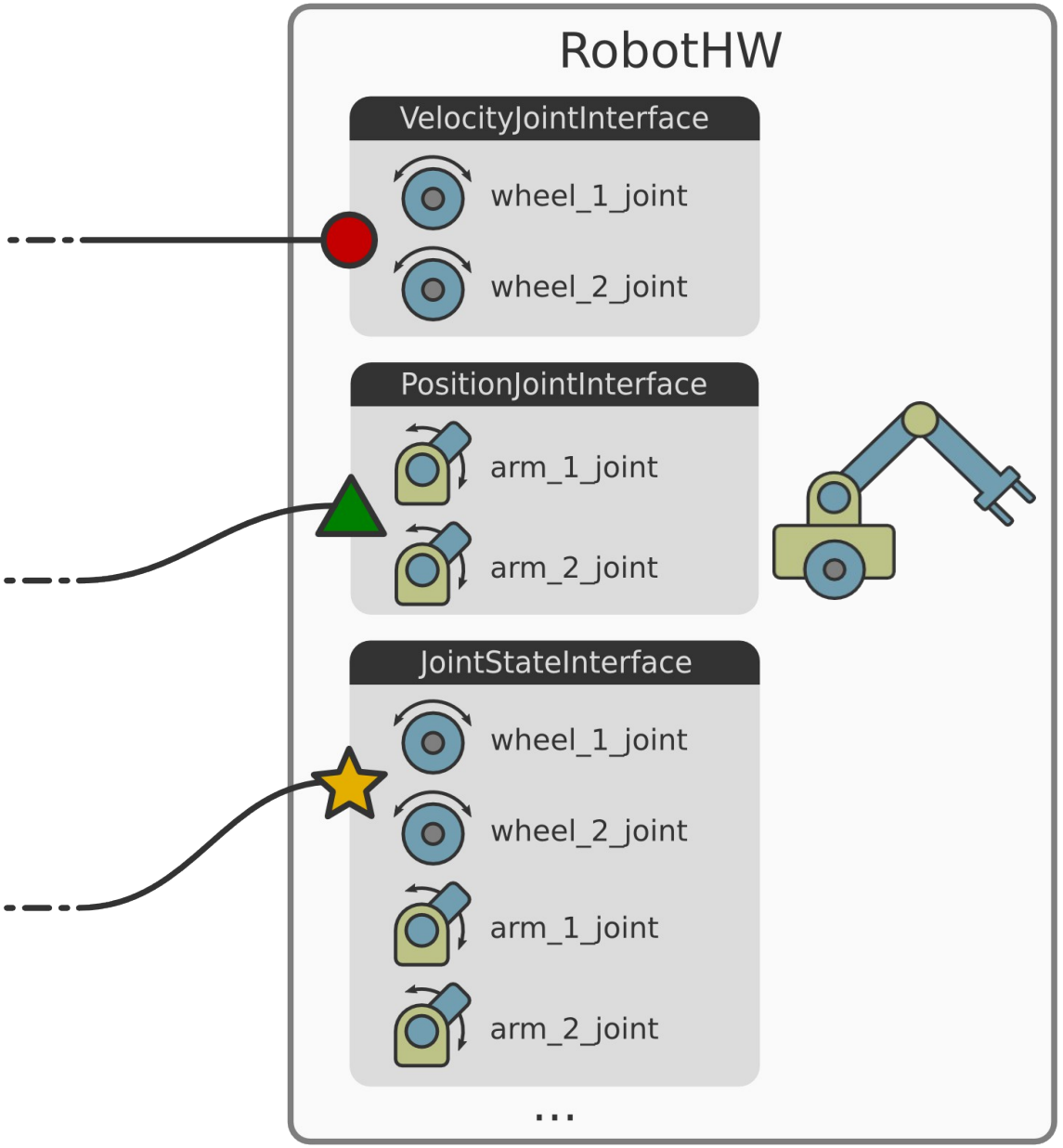


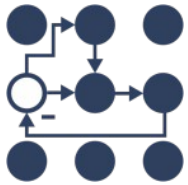
Setting up a robot



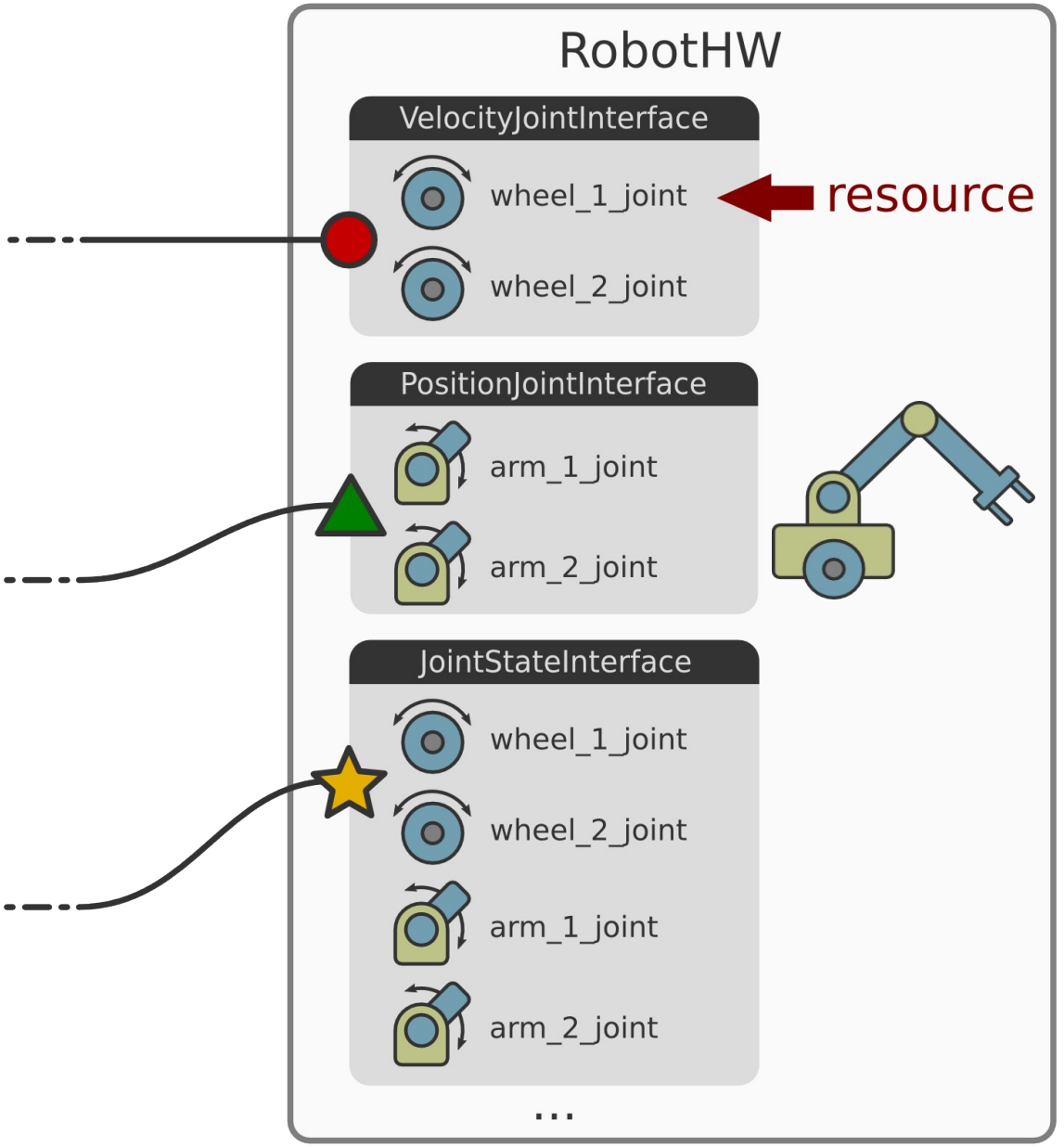


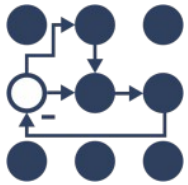
Setting up a robot



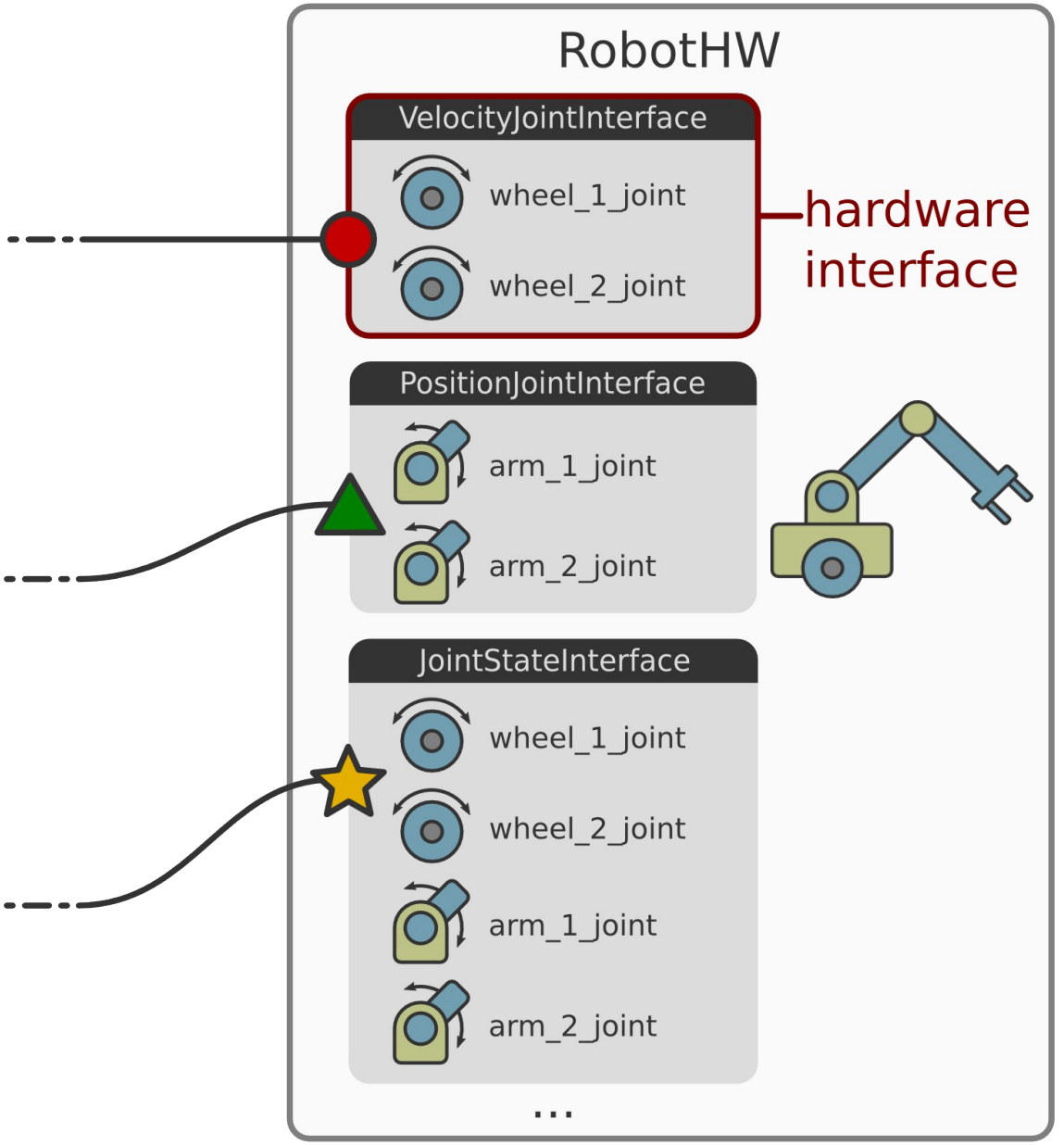


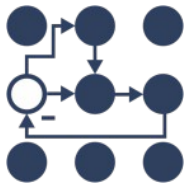
Setting up a robot



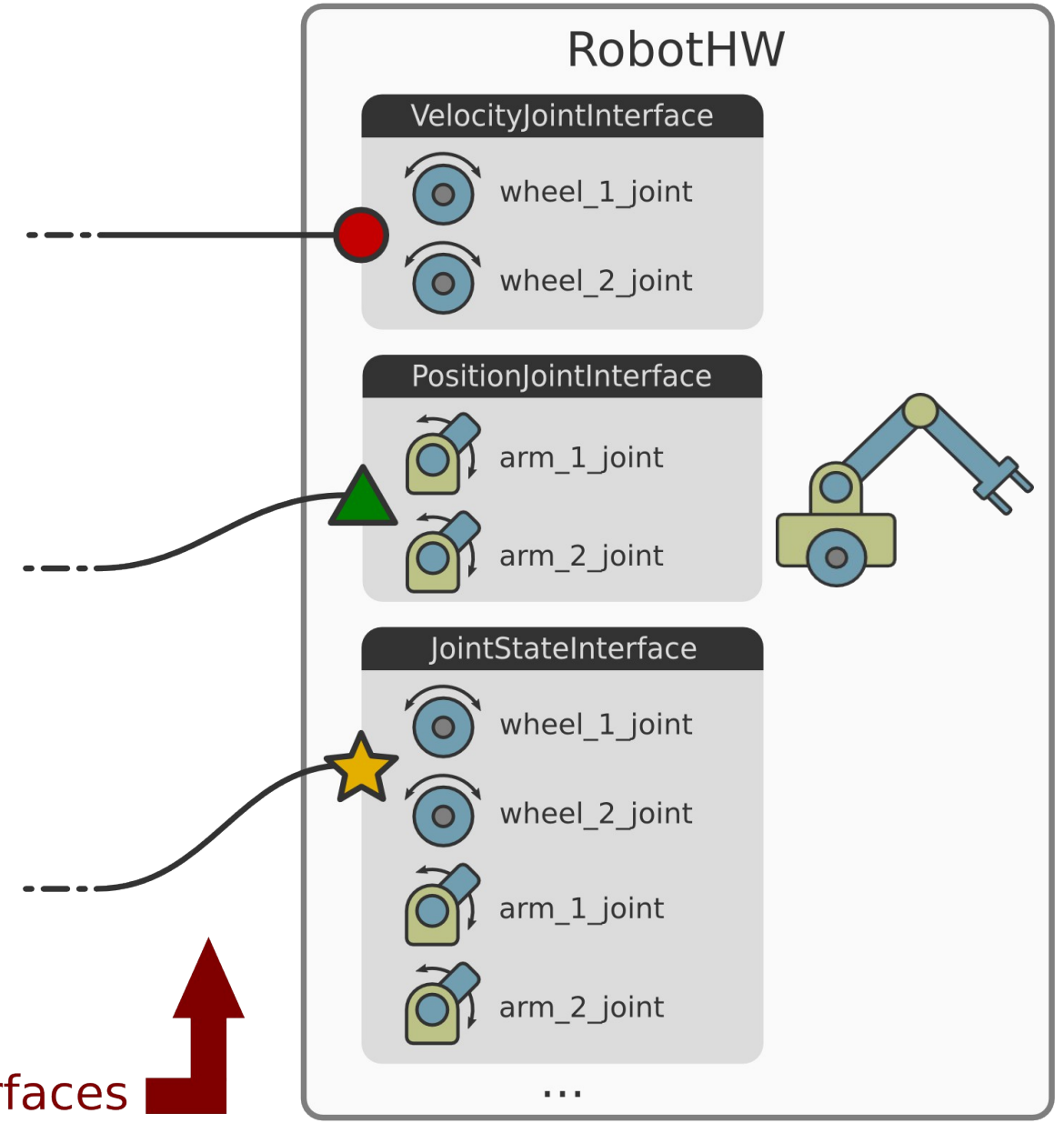


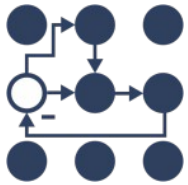
Setting up a robot



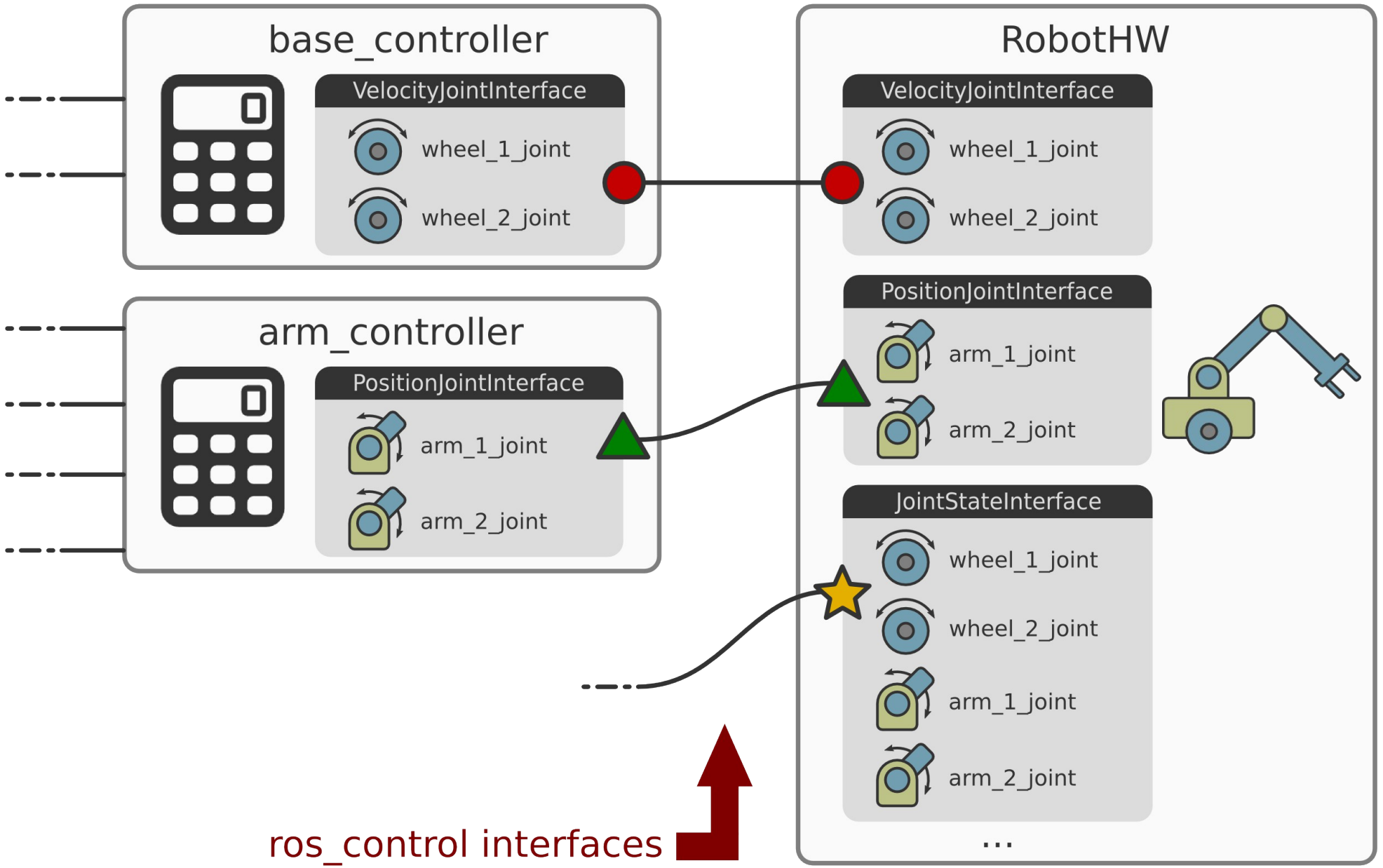


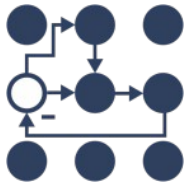
Setting up a robot



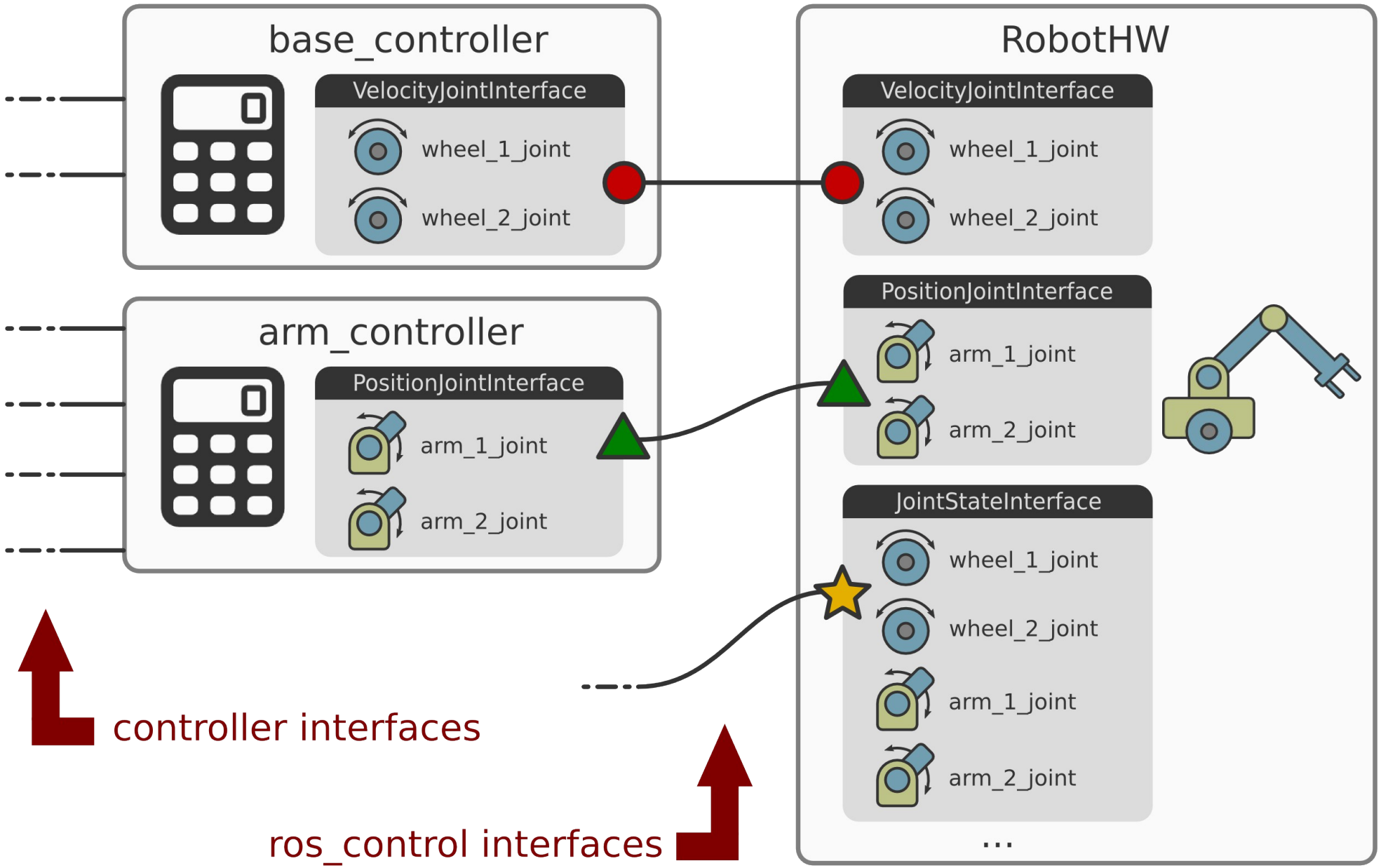


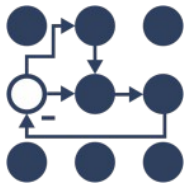
Setting up a robot



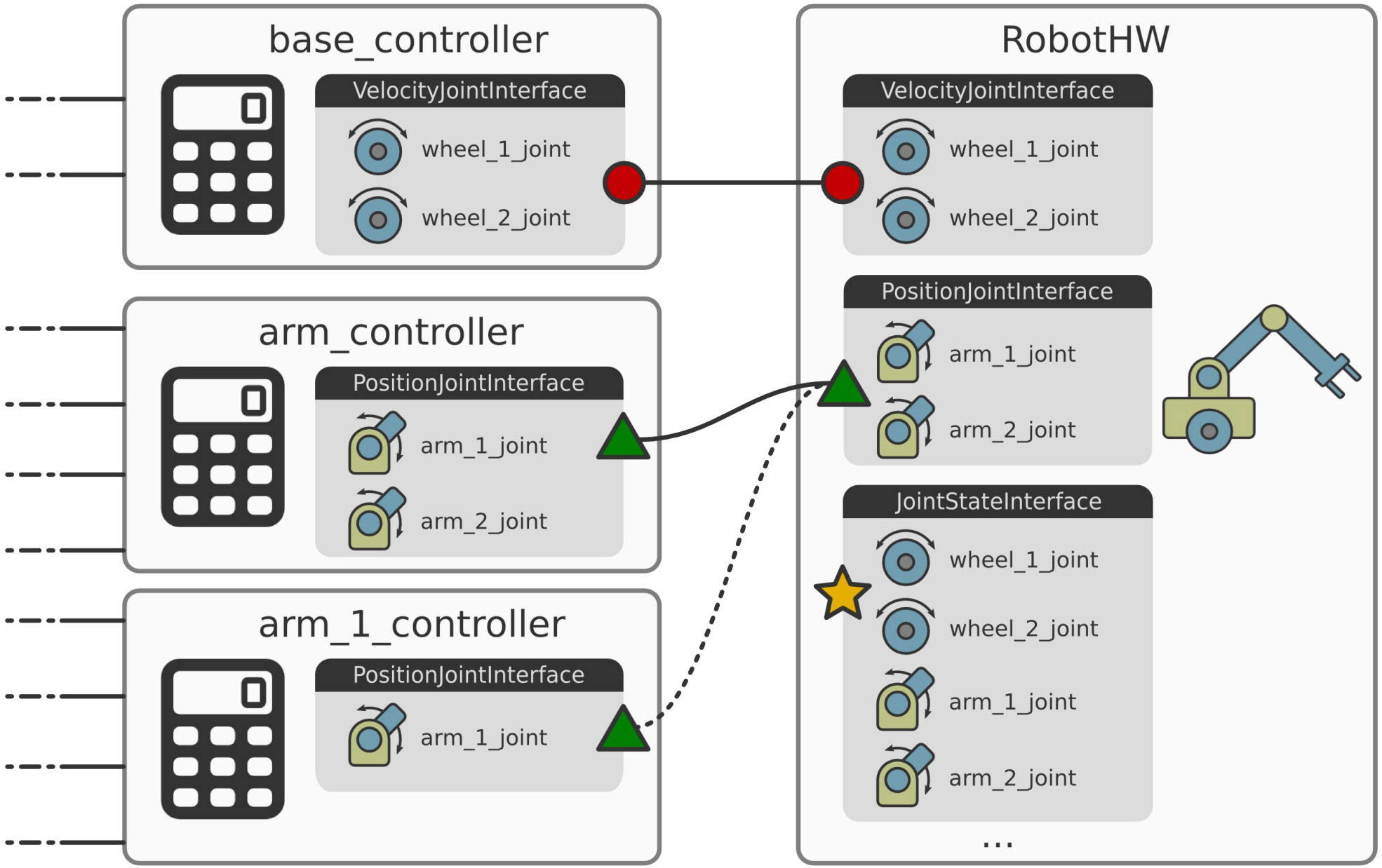


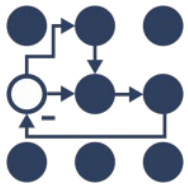
Setting up a robot



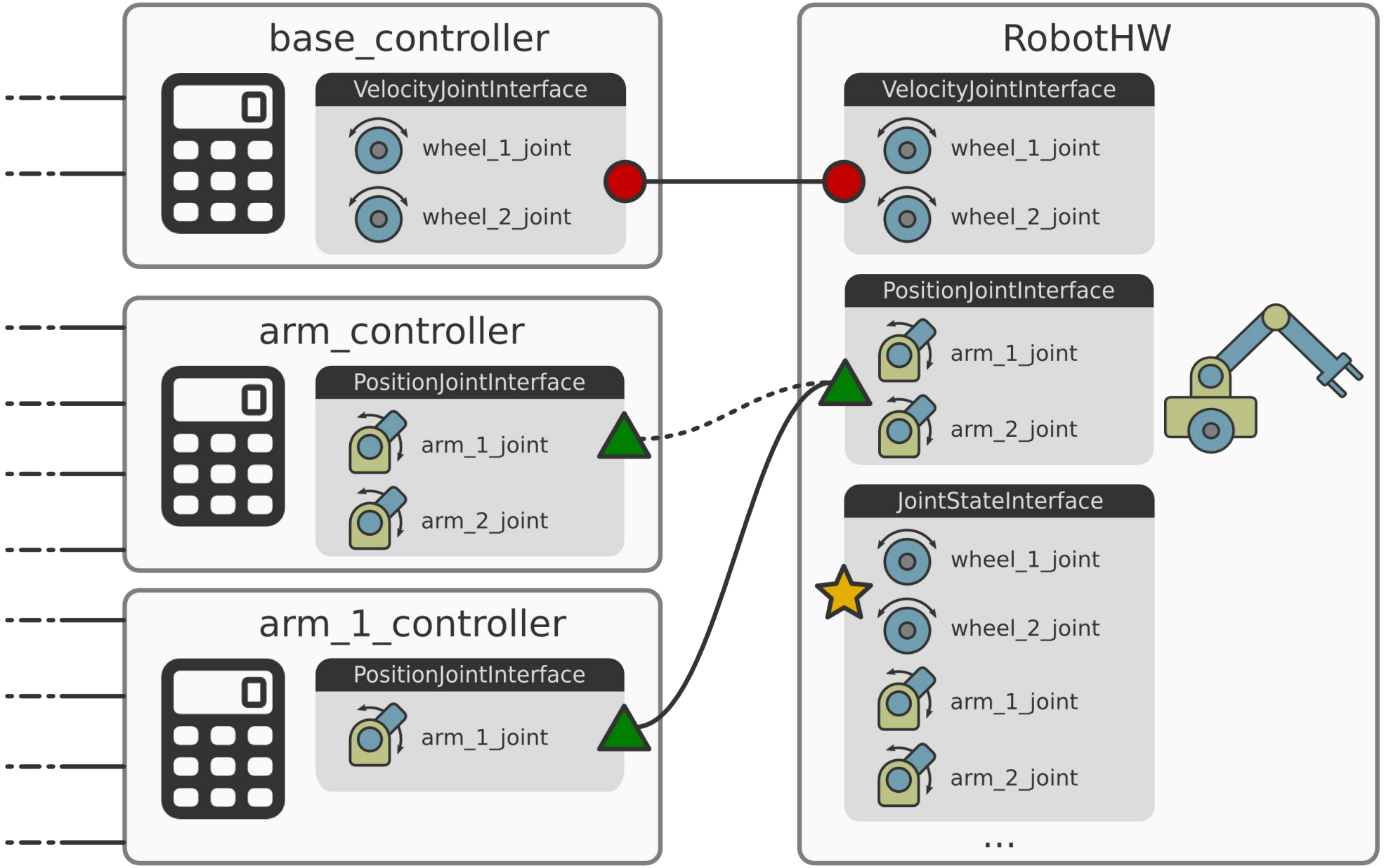


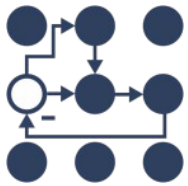
Setting up a robot



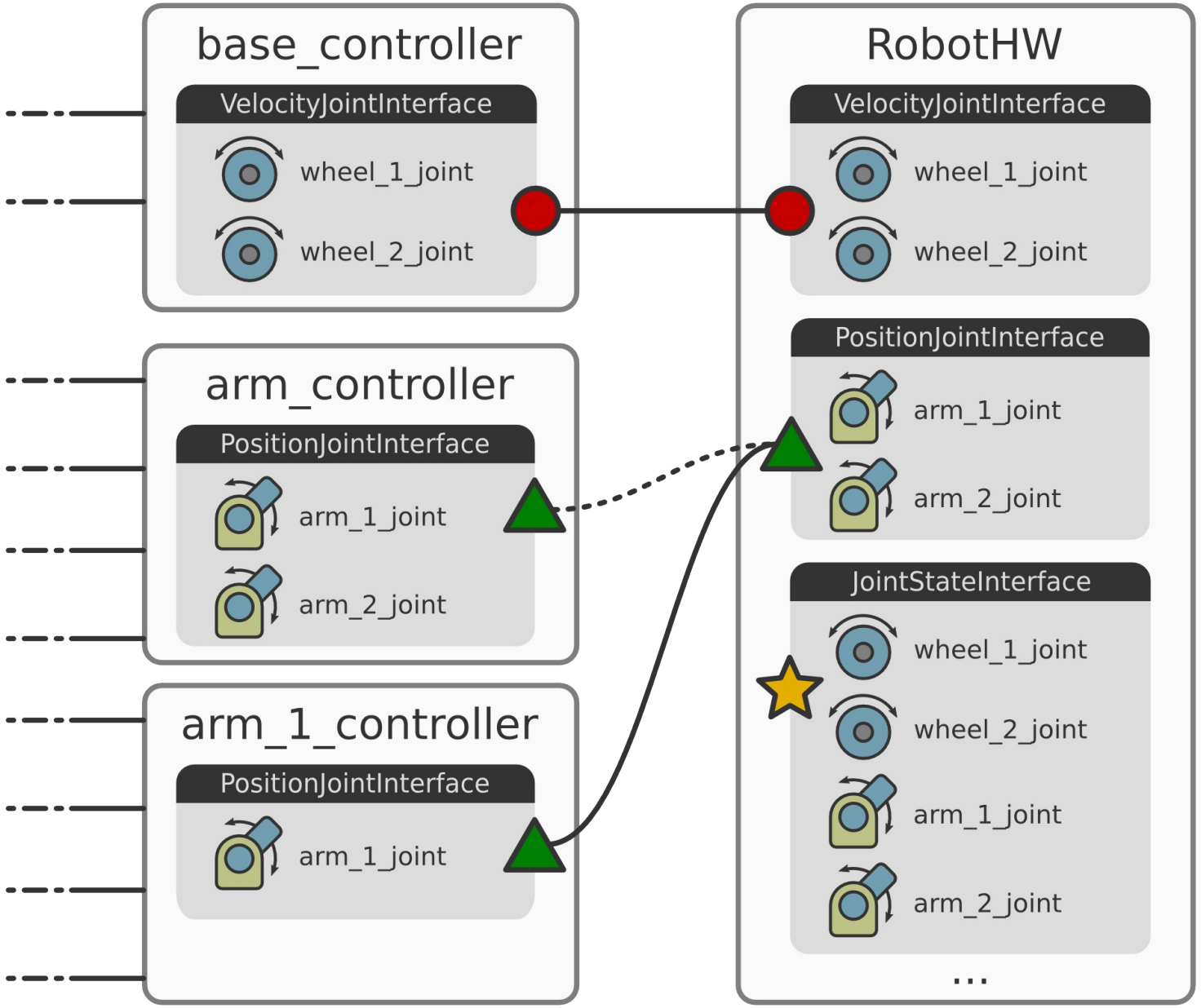


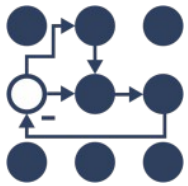
Setting up a robot



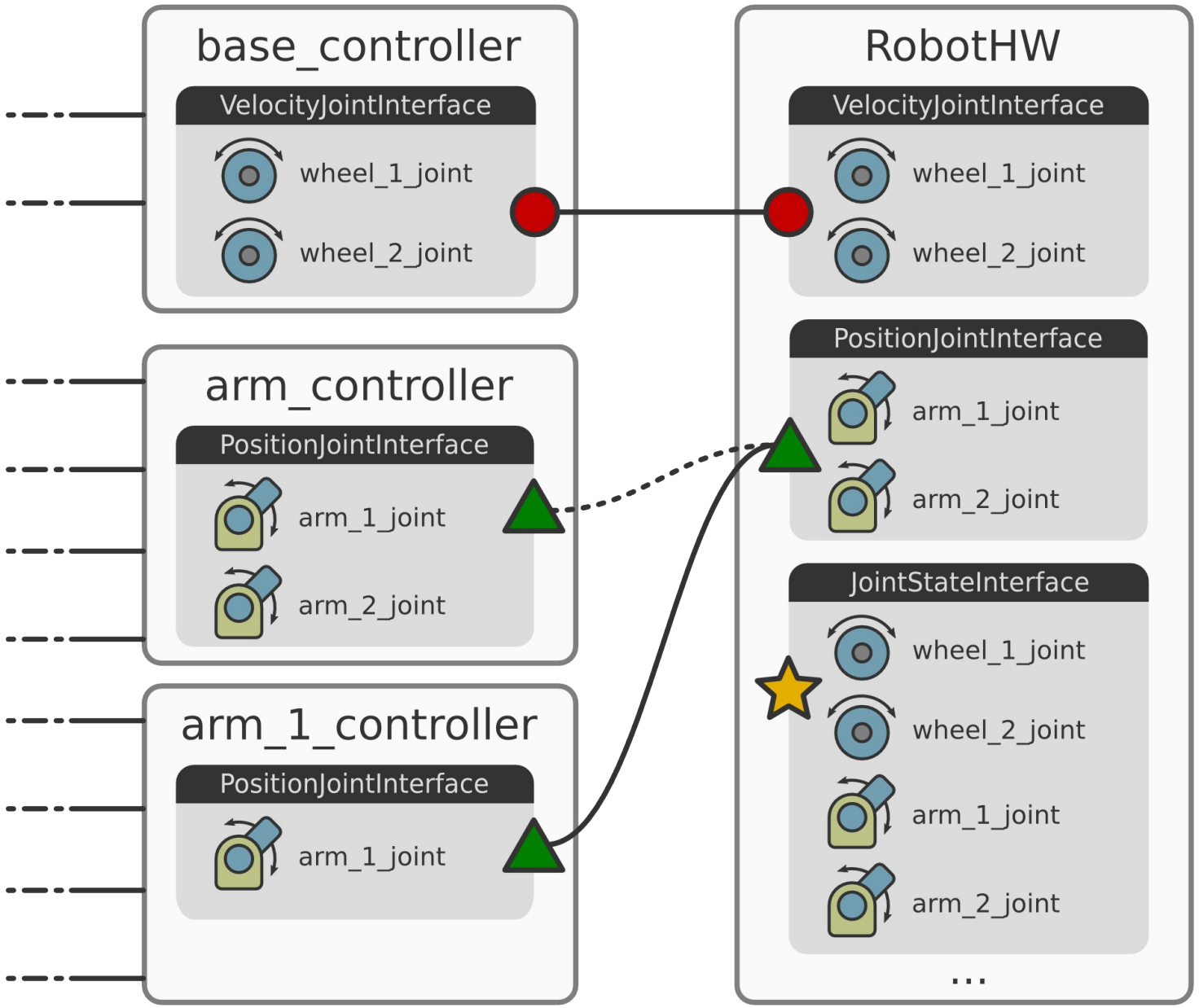


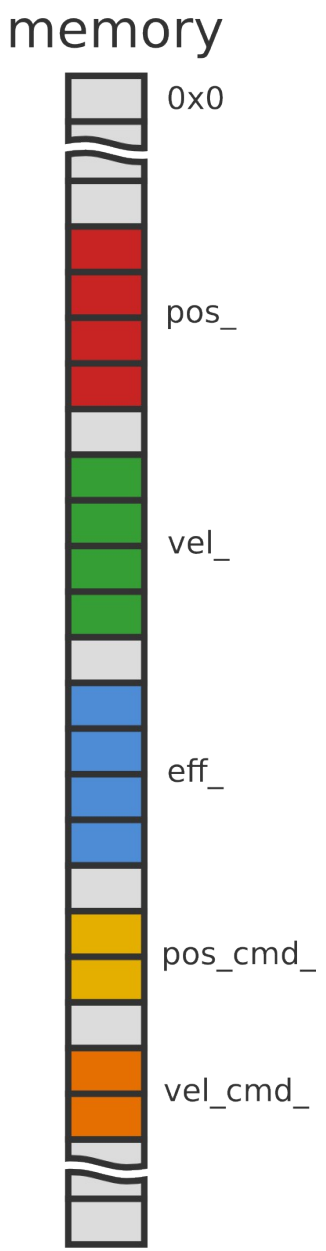
Setting up a robot

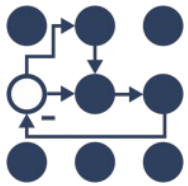




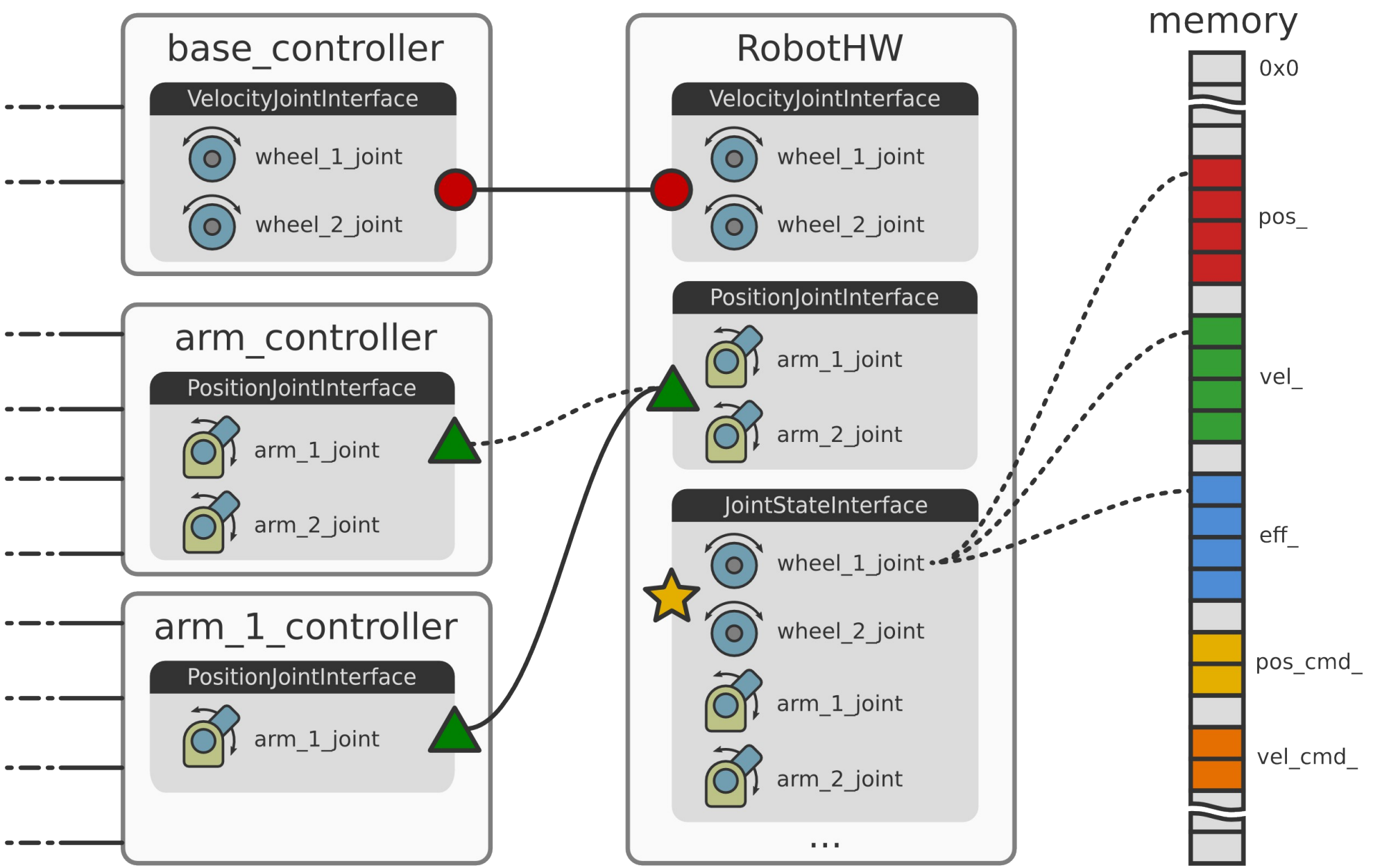
Setting up a robot

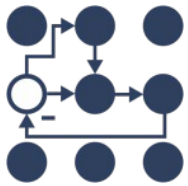




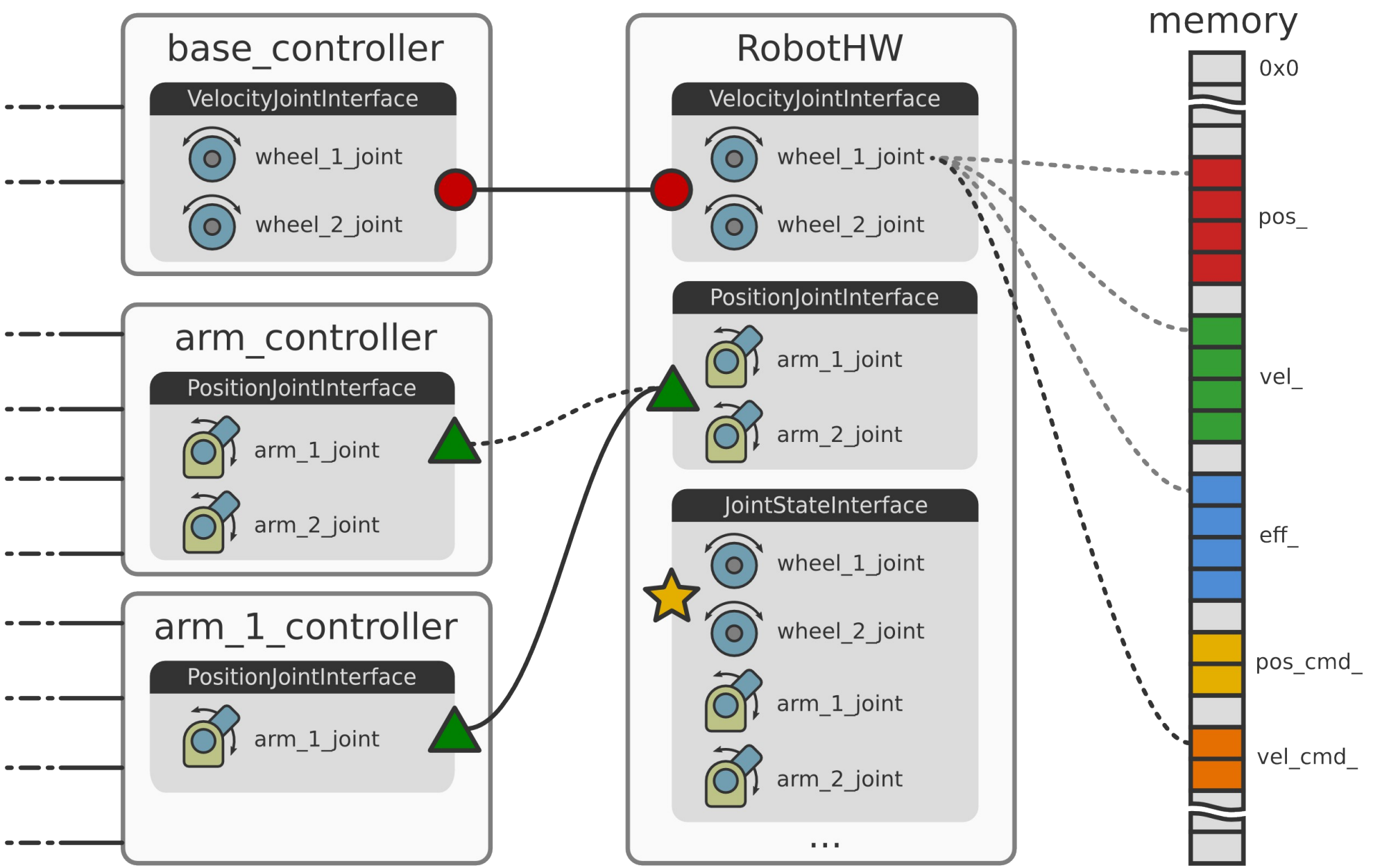


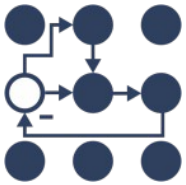
Setting up a robot





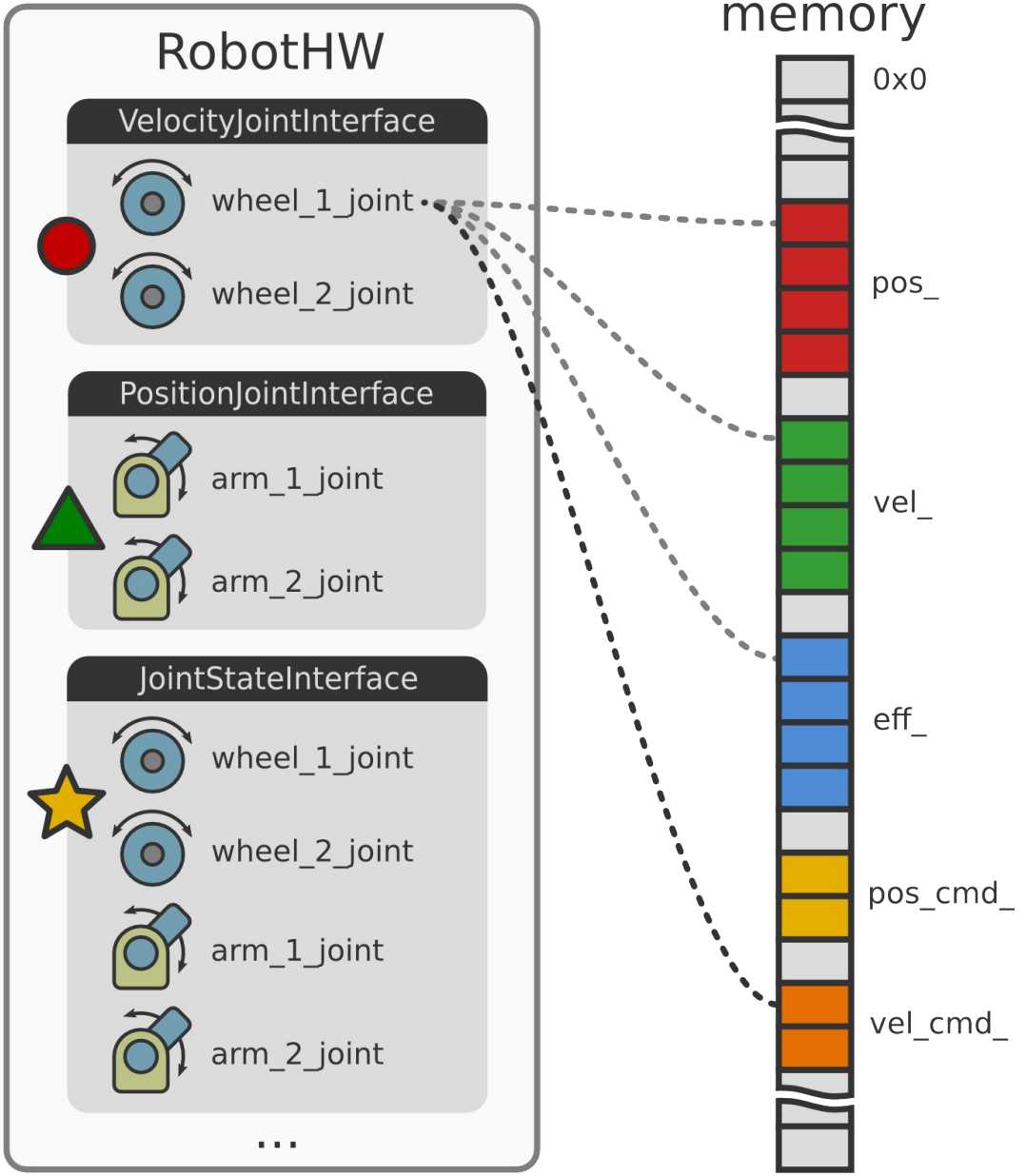
Setting up a robot

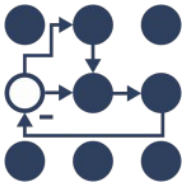




Setting up a robot

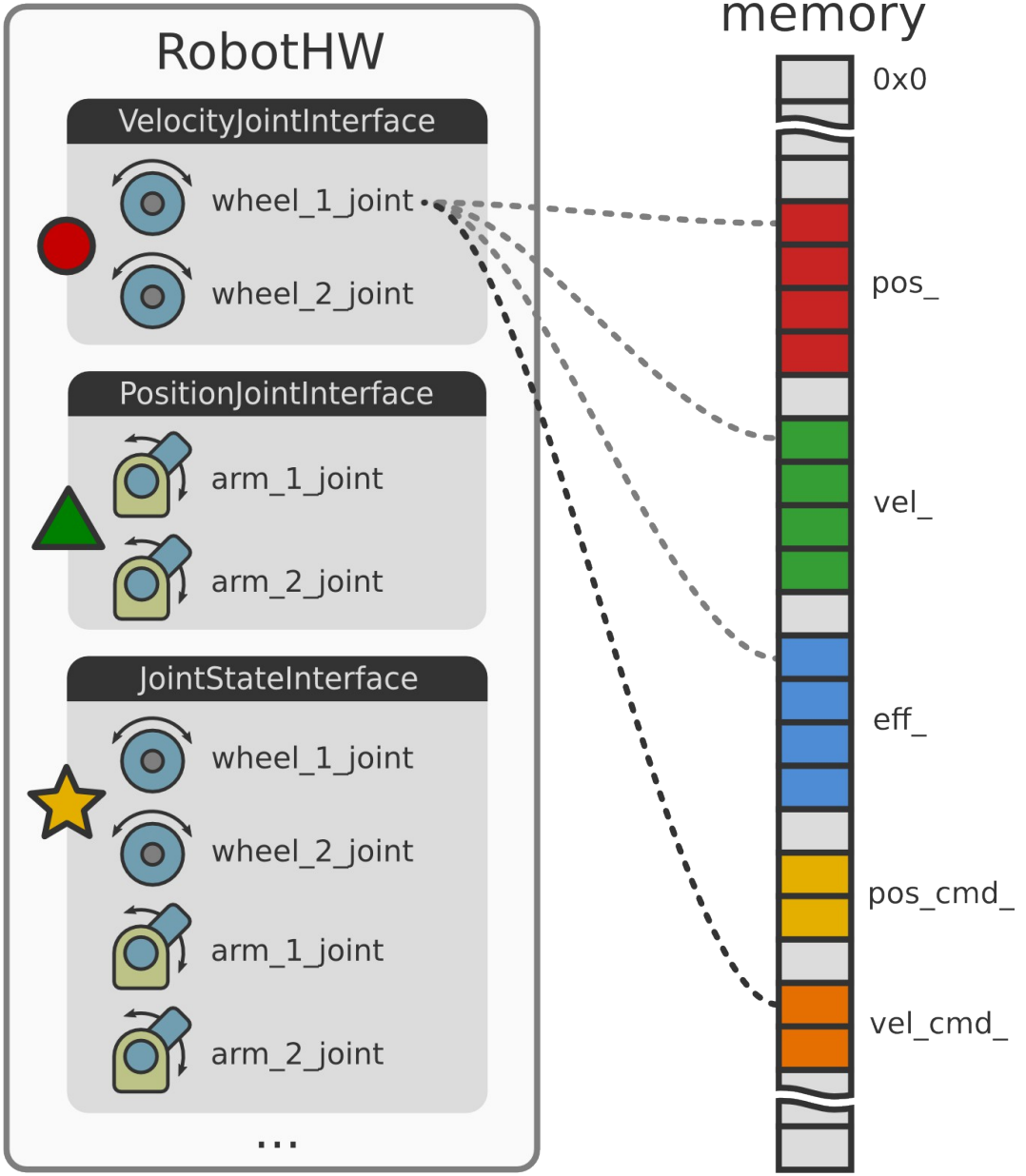
```
class MyRobot :
  public hardware_interface::RobotHW
{
public:
  MyRobot(); // Setup robot

  // Talk to HW
  void read();
  void write();

  // Reimplement only if needed
  virtual bool checkForConflict(...) const;
};
```

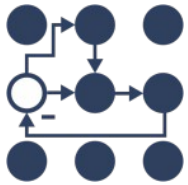




Setting up a robot

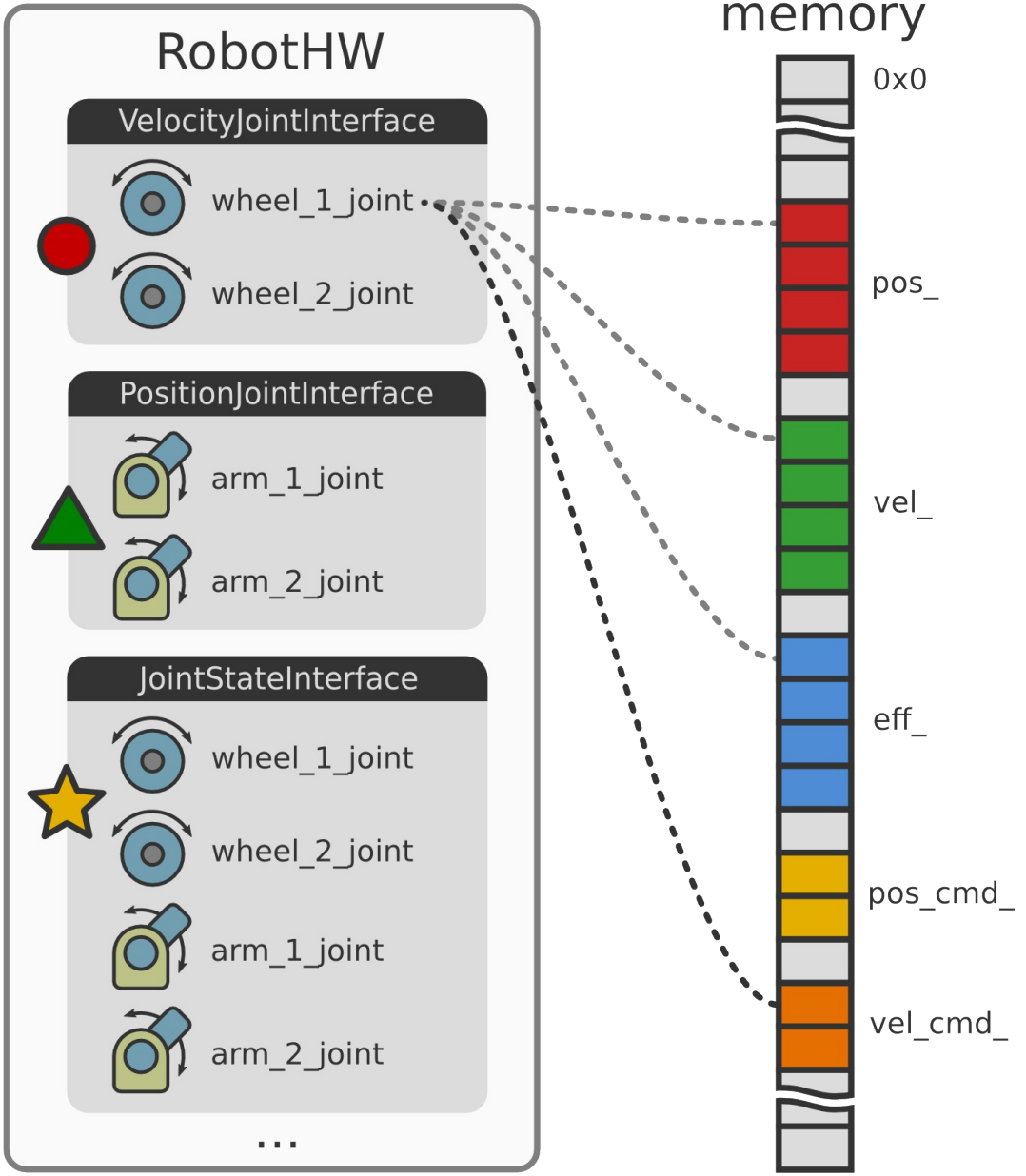
```
class MyRobot :
  public hardware_interface::RobotHW
{
public:
  MyRobot(); // Setup robot

  // Talk to HW
  void read();
  void write();

  // Reimplement only if needed
  virtual bool checkForConflict(...) const;
};
```

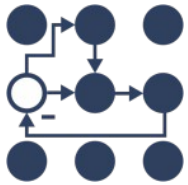




Setting up a robot

```
class MyRobot :
  public hardware_interface::RobotHW
{
public:
  MyRobot(); // Setup robot

  // Talk to HW
  void read();
  void write();

  // Reimplement only if needed
  virtual bool checkForConflict(...) const;
};
```

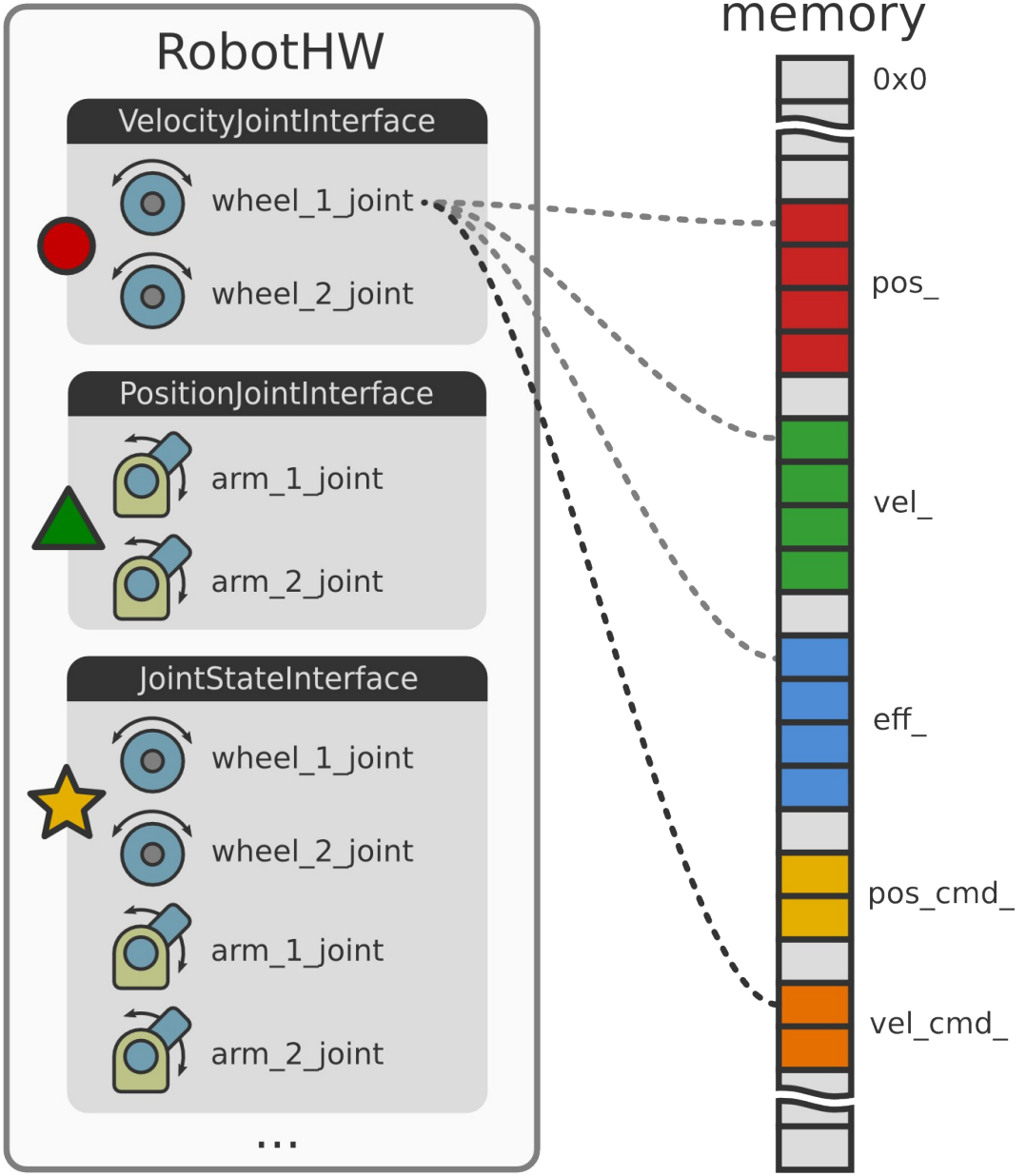


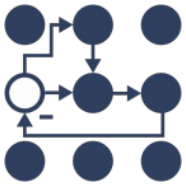


Setting up a robot

```
class MyRobot :
  public hardware_interface::RobotHW
{
public:
  MyRobot(); // Setup robot

  // Talk to HW
  void read();
  void write();

  // Reimplement onl
  virtual bool checkForConflict(...) const;
};
```

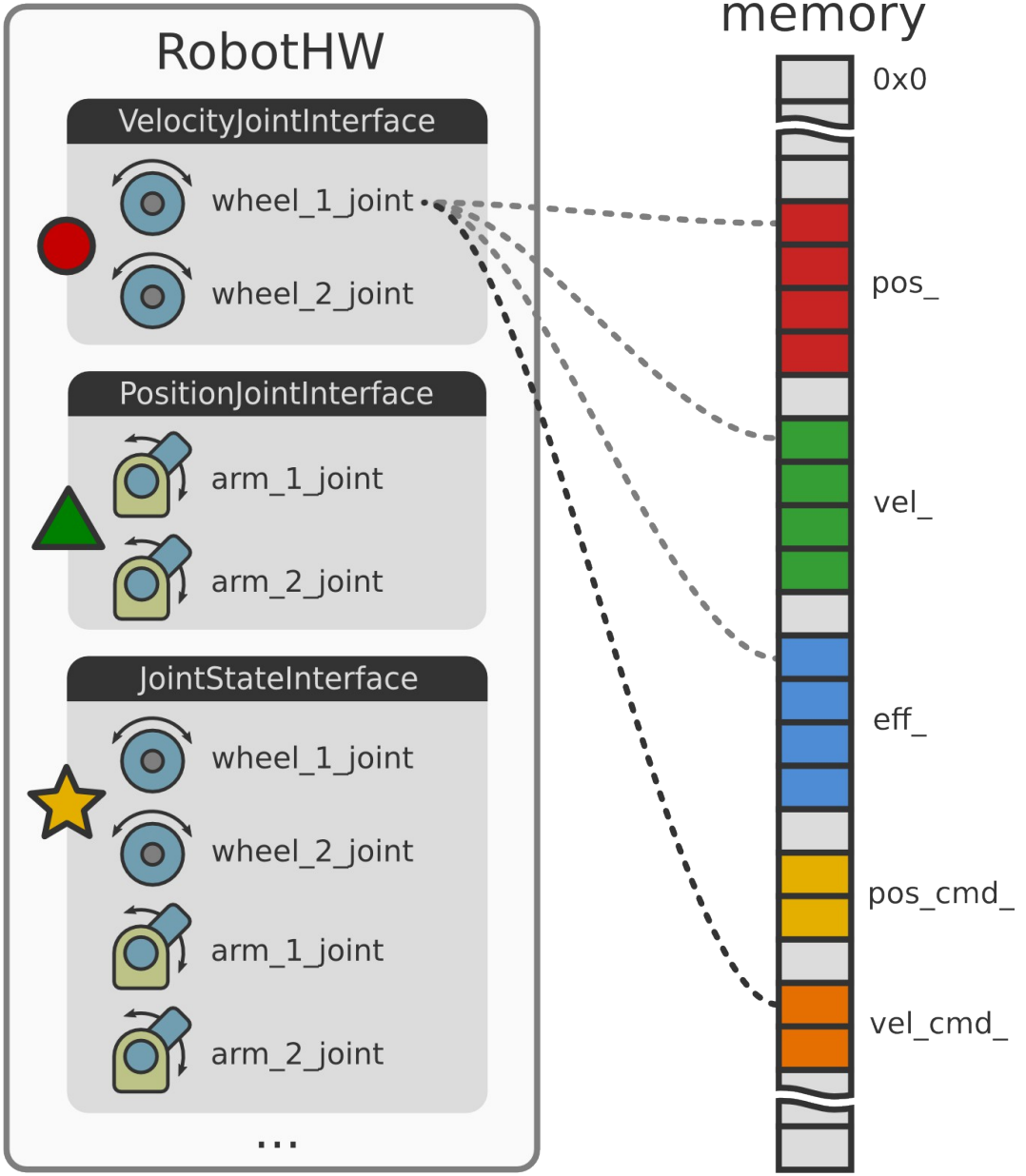
- ROS industrial
- rosserial
- SR RONEX
- custom
- ...

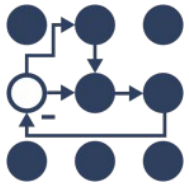




Setting up a robot

```
class MyRobot :
  public hardware_interface::RobotHW
{
public:
  MyRobot(); // Setup robot

  // Talk to HW
  void read();
  void write();

  // Reimplement only if needed
  virtual bool checkForConflict(...) const;
};
```

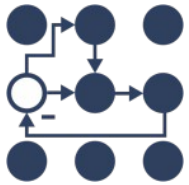




hardware_interface

Robot hardware abstraction

- **Software** representation of robot
- Abstracts **hardware** away
  - **Resource:** actuators, joints, sensors
  - **Interface:** Set of similar resources
  - **Robot:** Set of interfaces
- Handles **resource conflicts**
  - **Exclusive** ownership by default

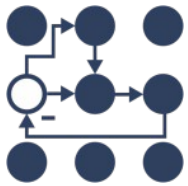


hardware_interface

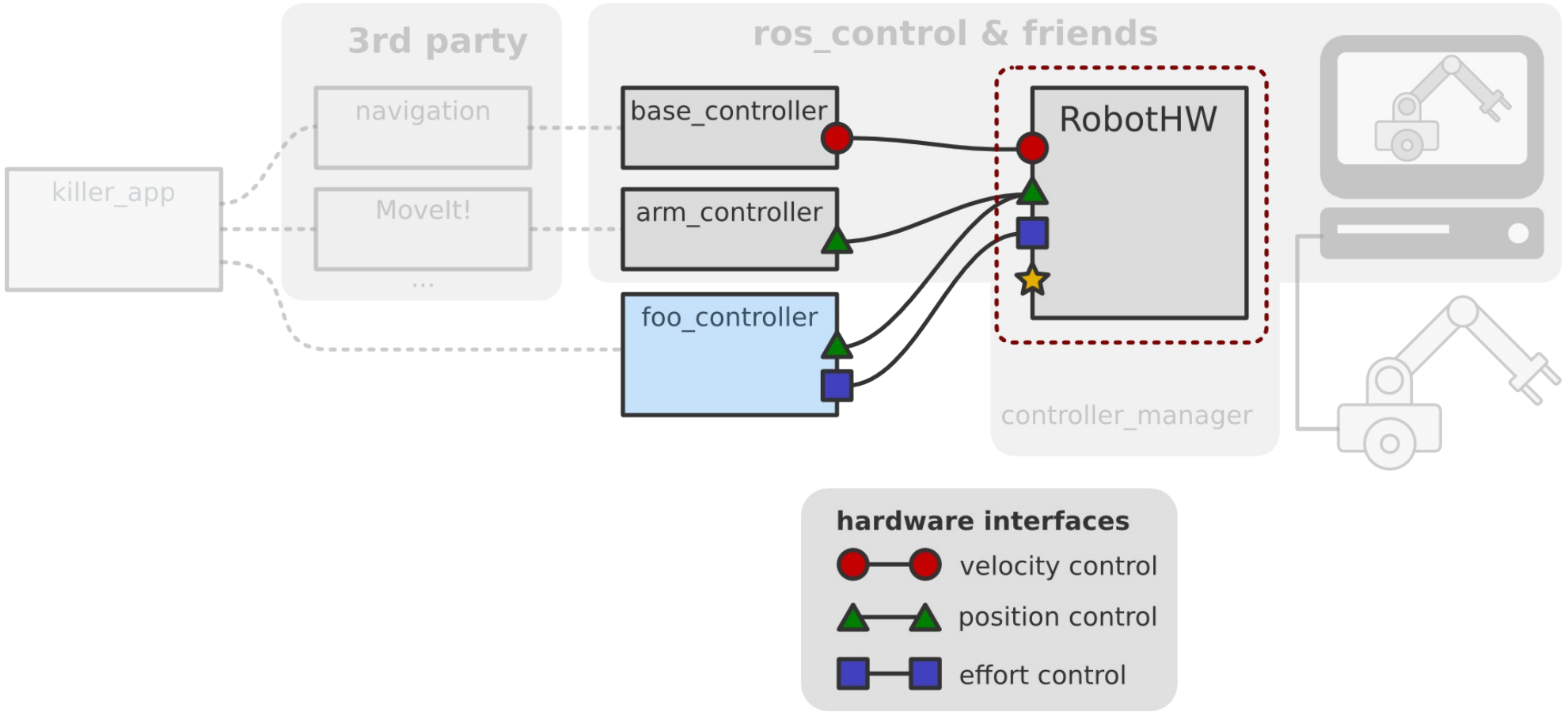
Resources and interfaces

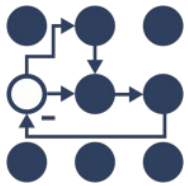
- Read-only
  - Joint state*
  - IMU
  - Force-torque sensor
- Read-write
  - Position joint*
  - Velocity joint*
  - Effort joint*

* Equivalent interfaces exist for actuators

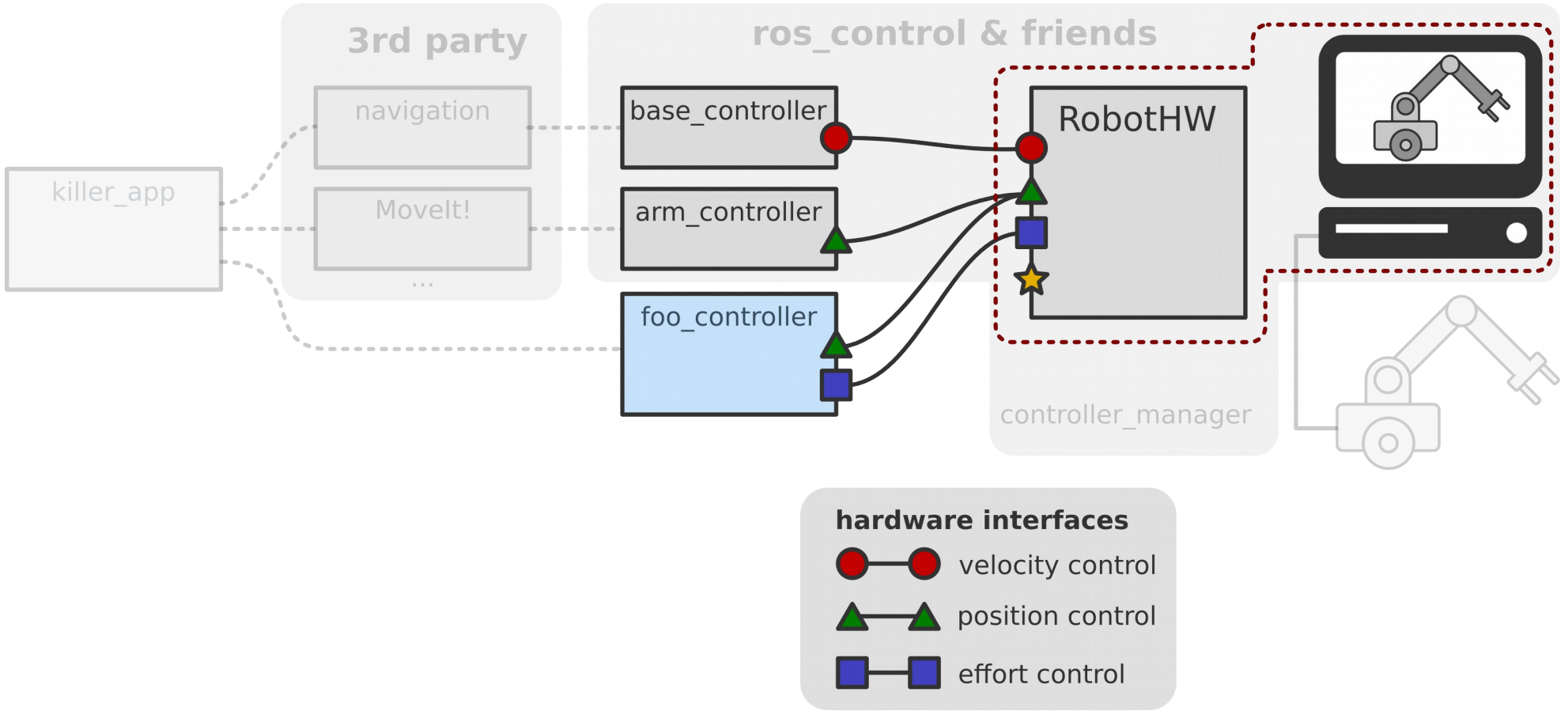


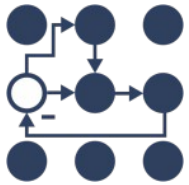
Setting up a robot





Setting up a robot





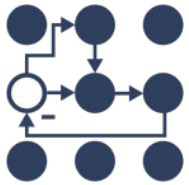
gazebo_ros_control

- Lives **outside** ros-controls repos

   ros-simulation/**gazebo_ros_pkgs**

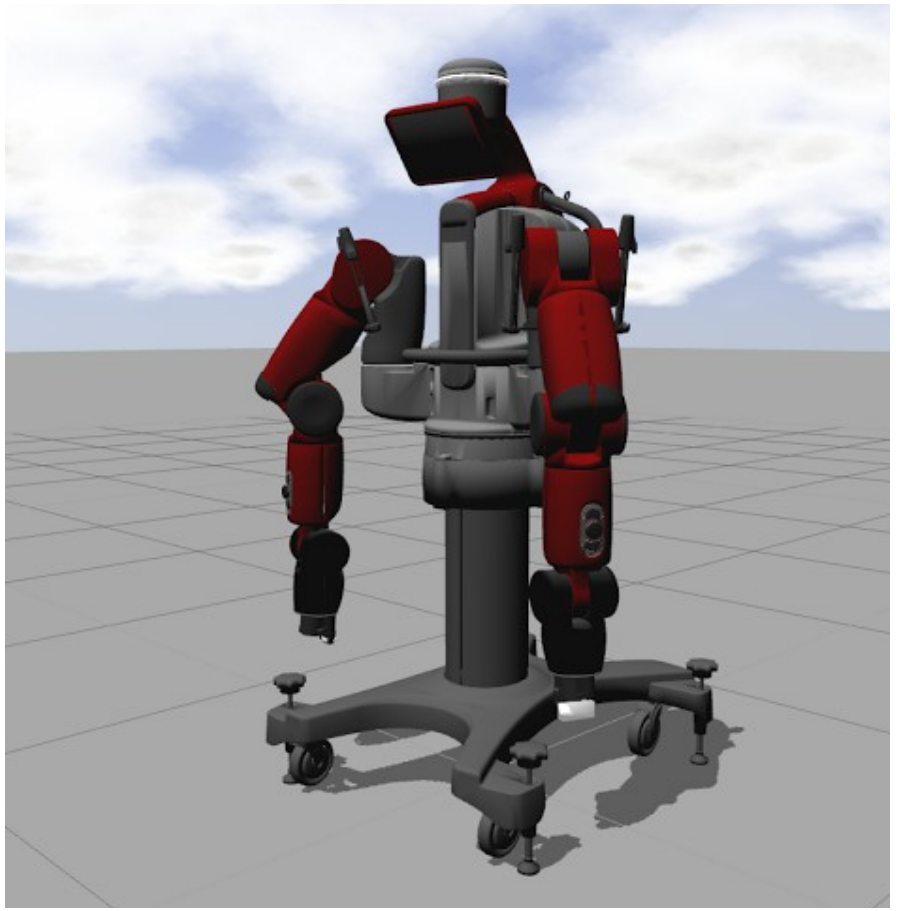
- **Gazebo** plugin for **ros_control**
  - **Default plugin:**
    - Populates **joint interfaces** from **URDF**
    - Reads **transmission** and **joint limits** specs
  - **Custom plugin:** Up to you

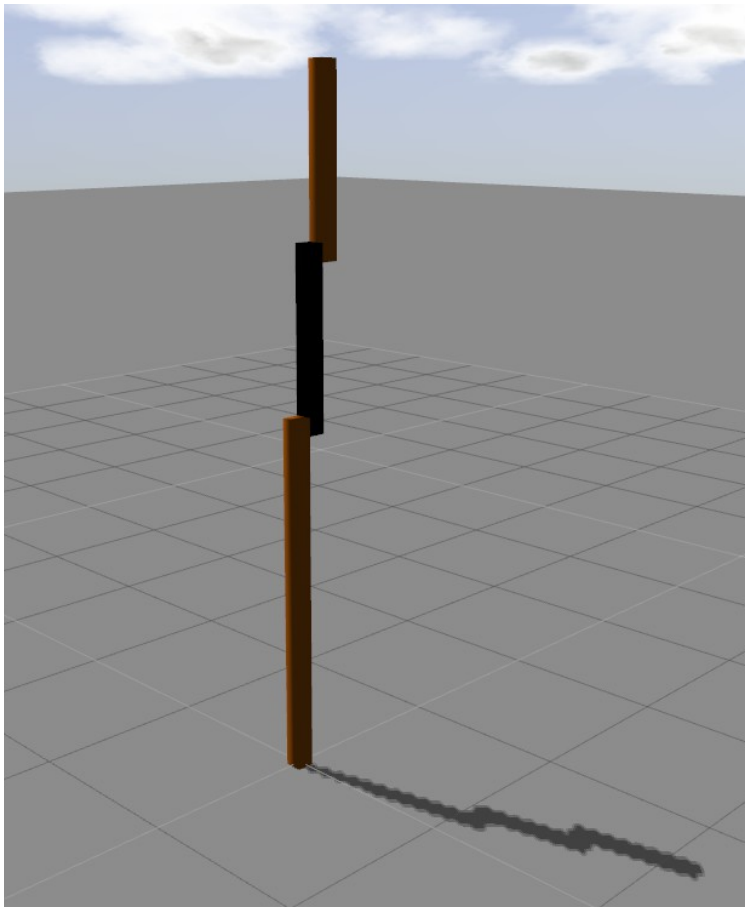
```
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/my_robot</robotNamespace>
  </plugin>
</gazebo>
```

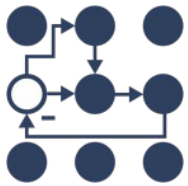


gazebo_ros_control

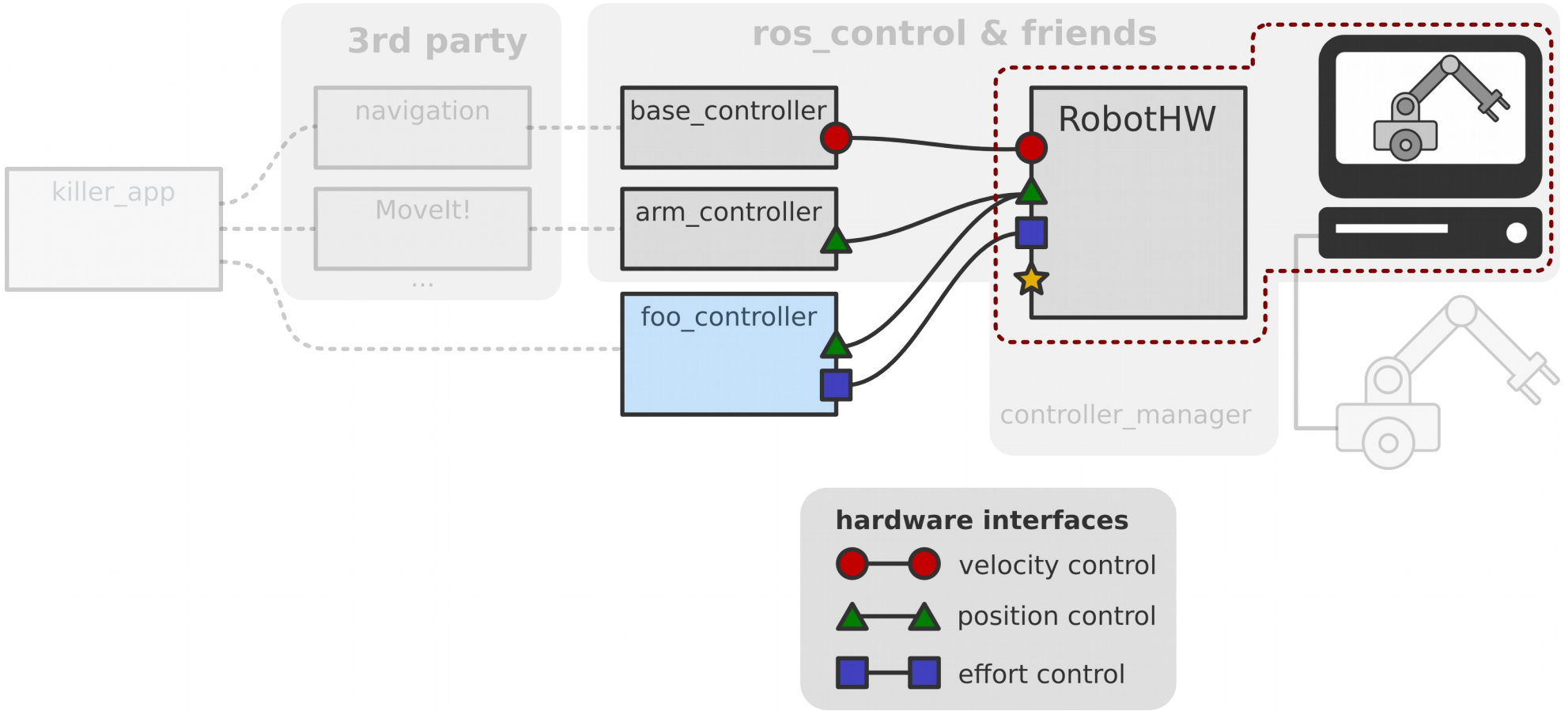
- Test ros_control without coding a RobotHW!
  - Only setup configuration files

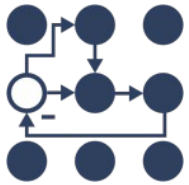






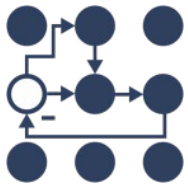
Setting up a robot



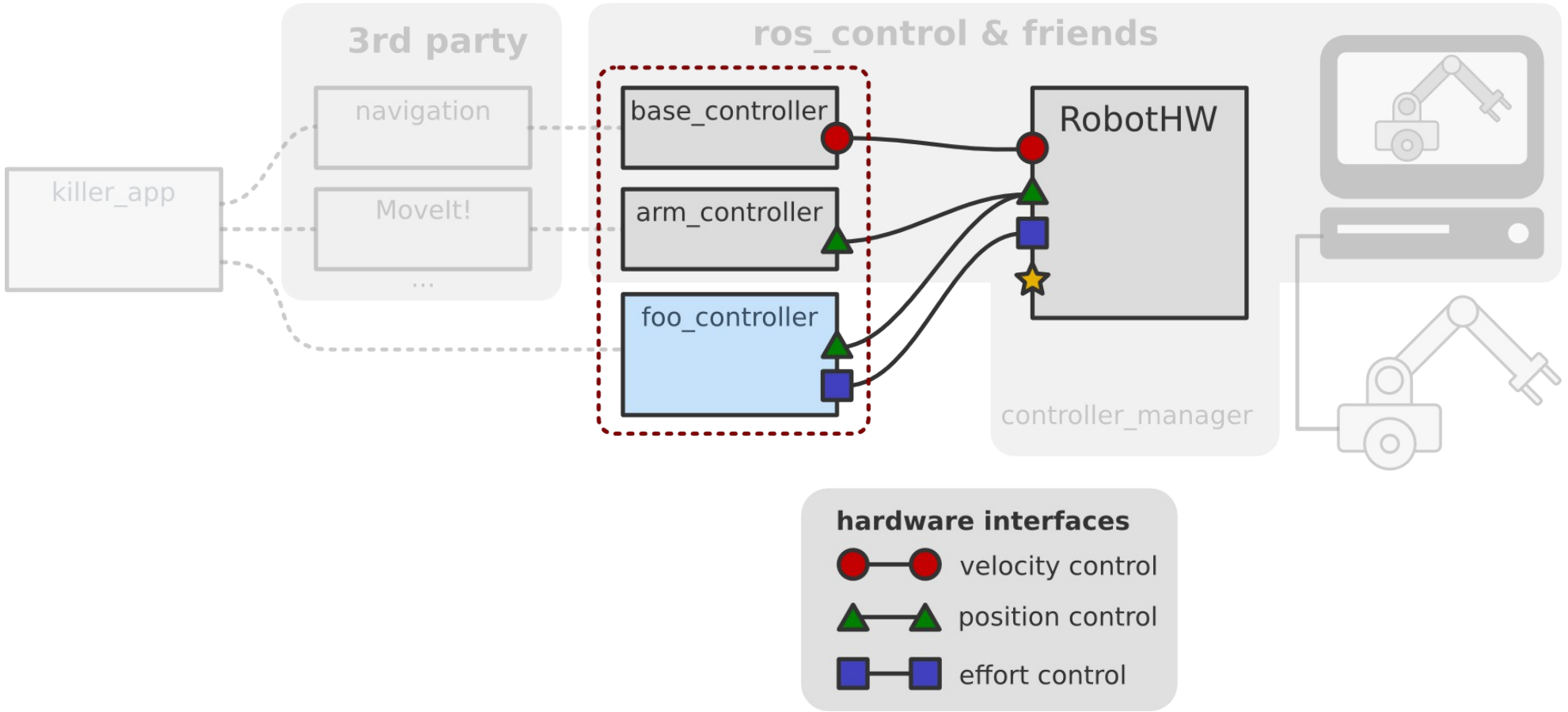


ROS control – an overview

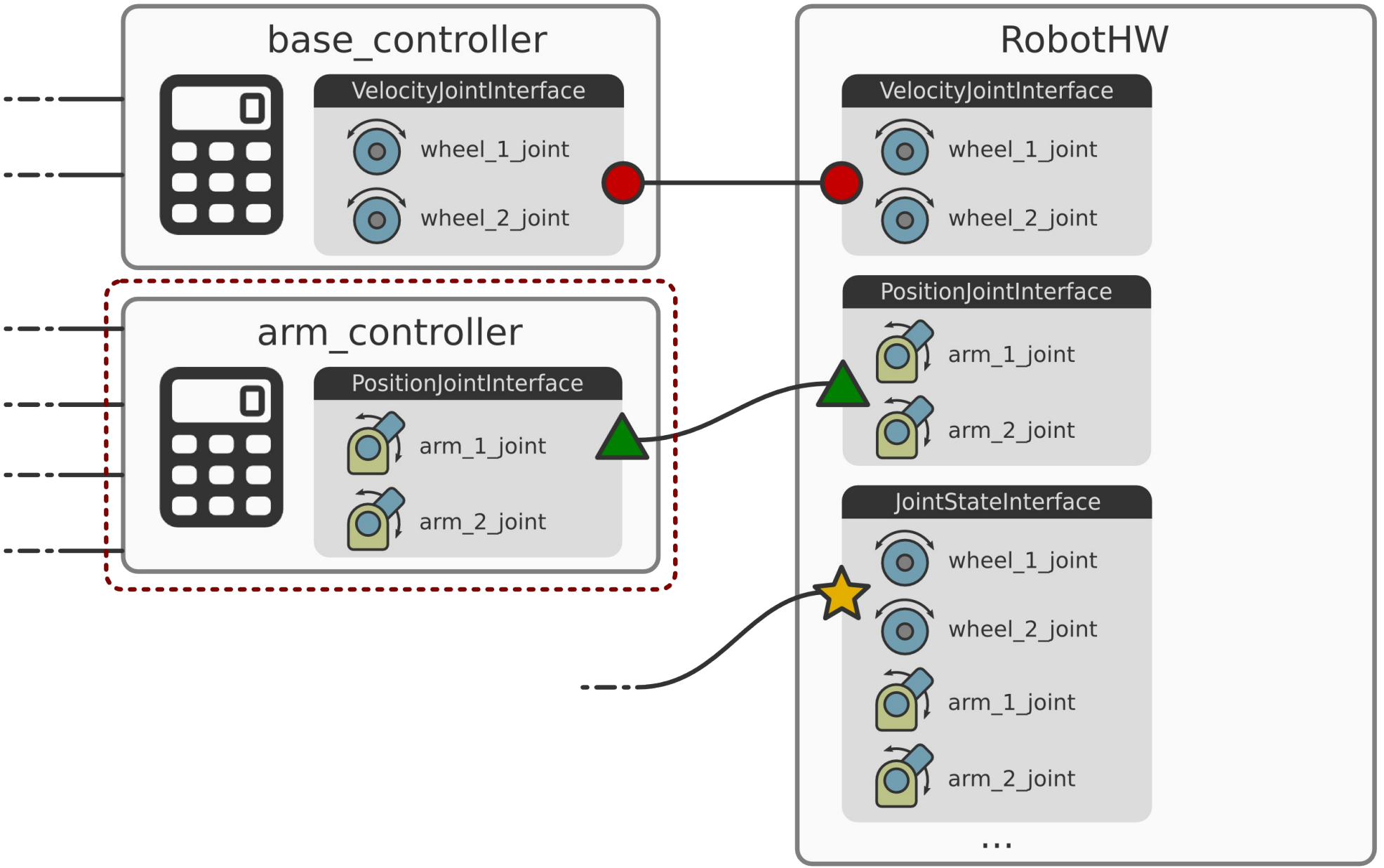
- Big picture and goals
- ROS control & friends
  - Setting up a robot
  - **Controllers**
  - The control loop
- Demo
- Robots using ROS control

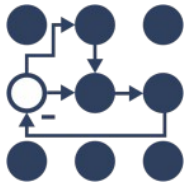


Controllers

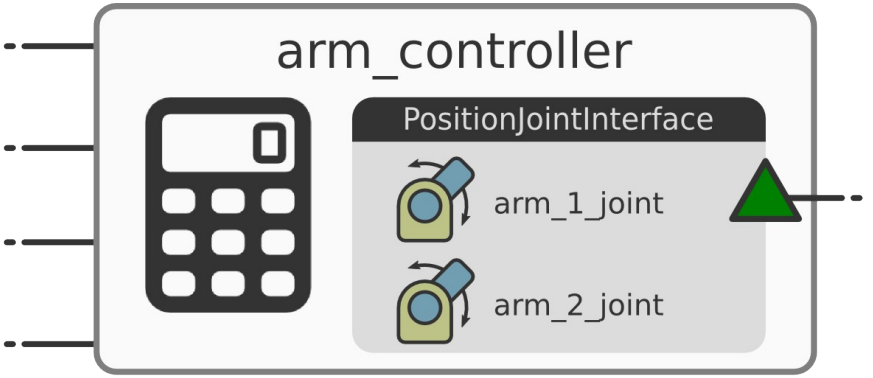


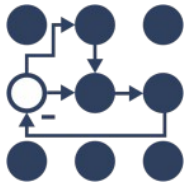
Controllers



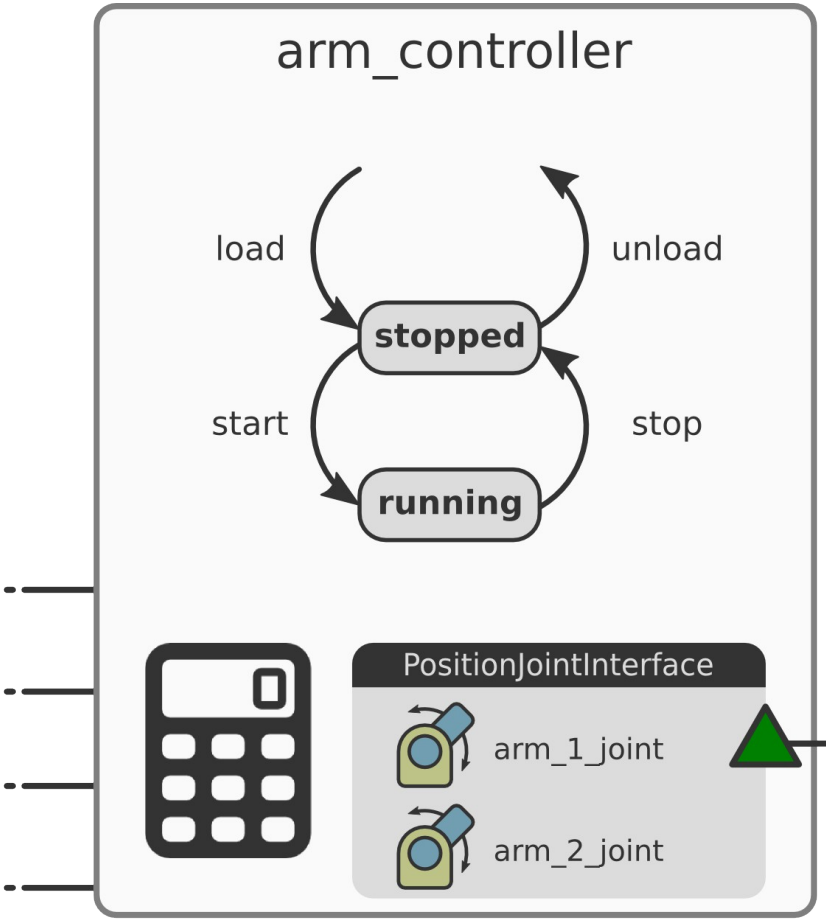


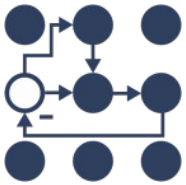
Controllers



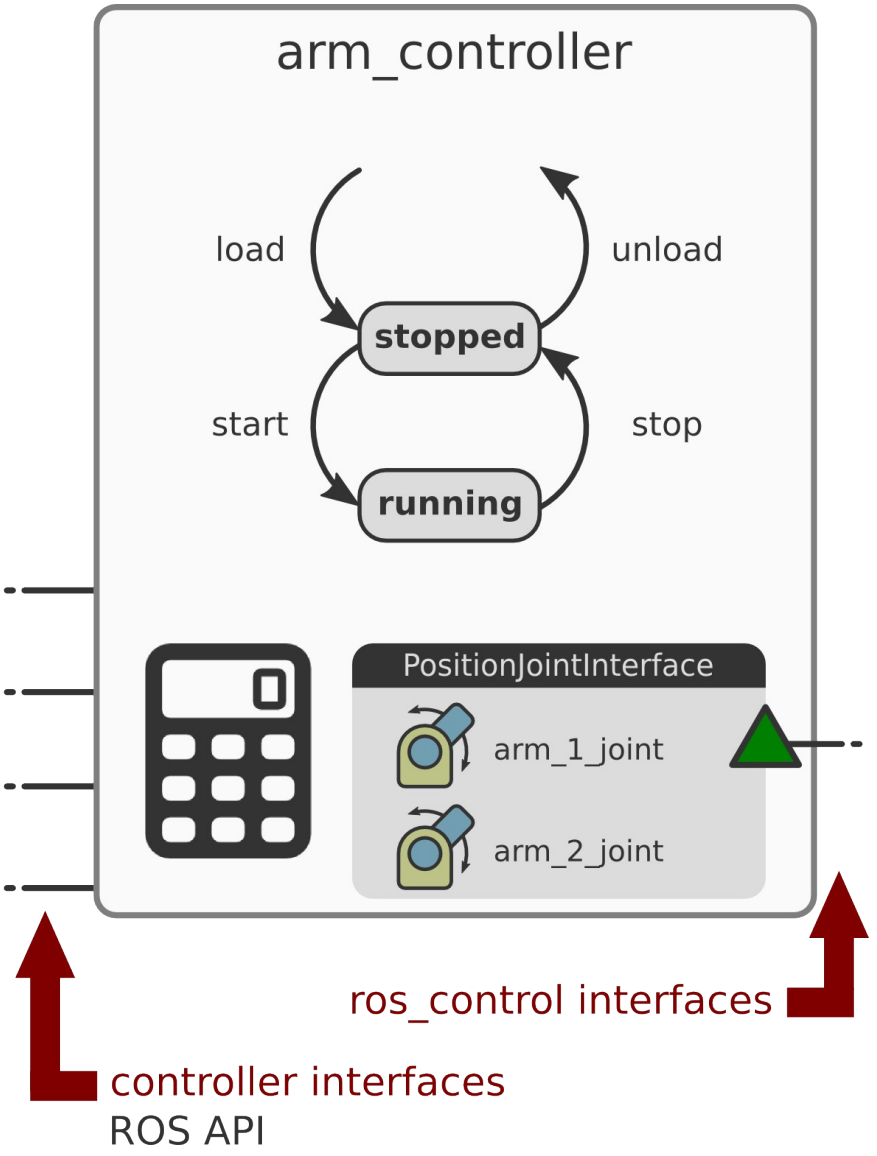


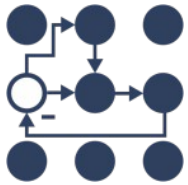
Controllers



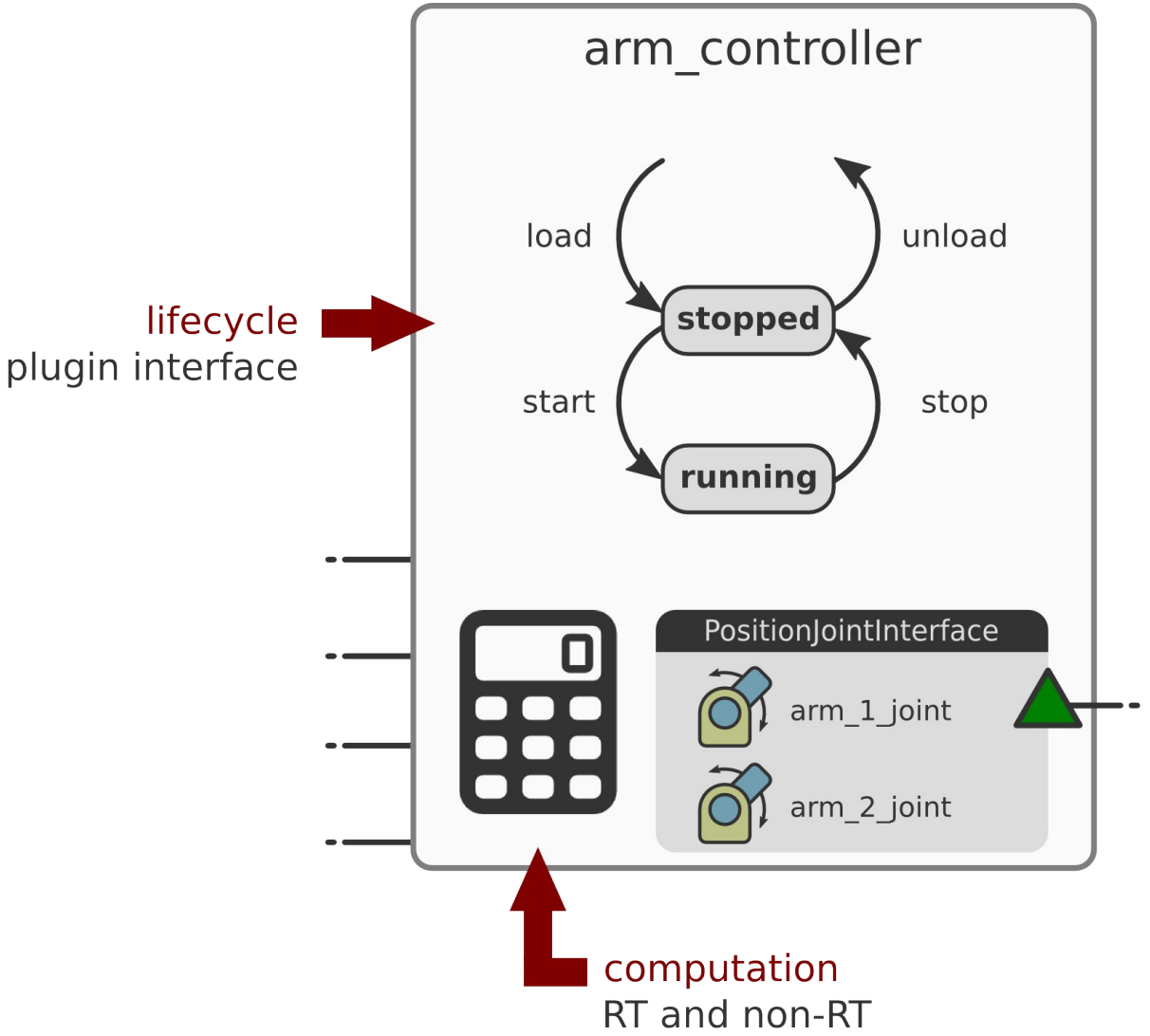


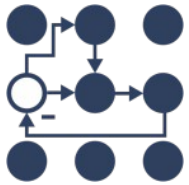
Controllers





Controllers

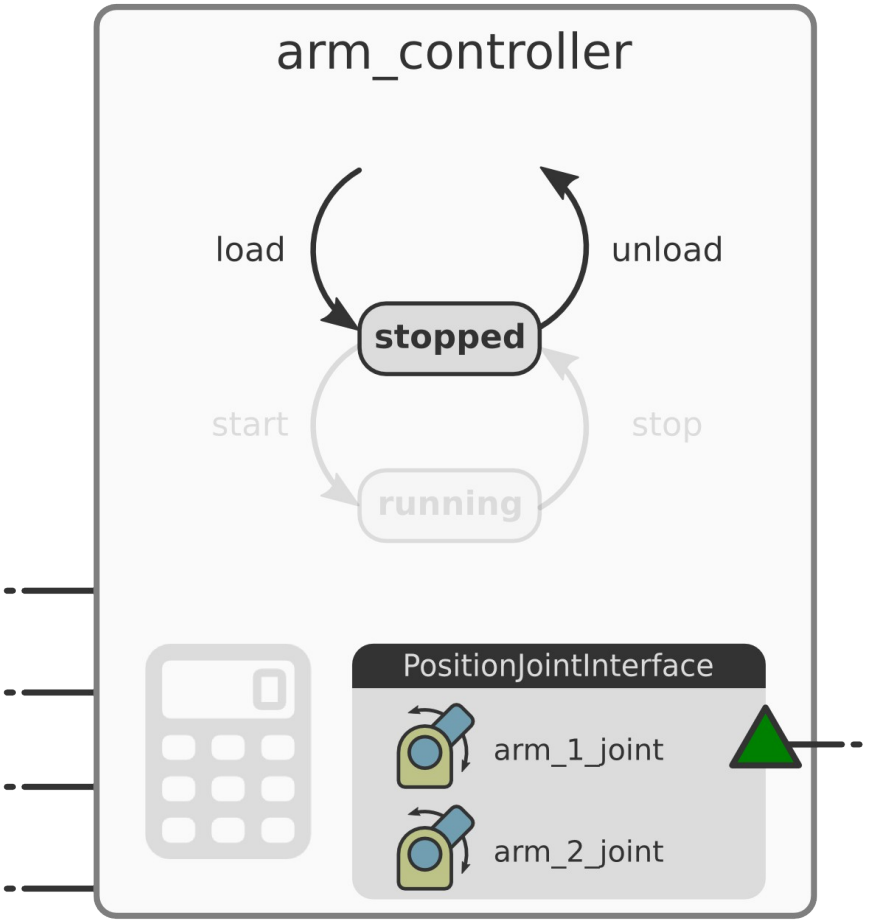




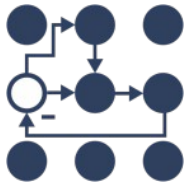
Controllers

Non real-time operations

- **load**
  - load + initialize plugin
  - check requisites (can fail)
    - hardware resource **existence***
    - configuration
  - setup ROS interfaces
- **unload**
  - destroy + unload plugin



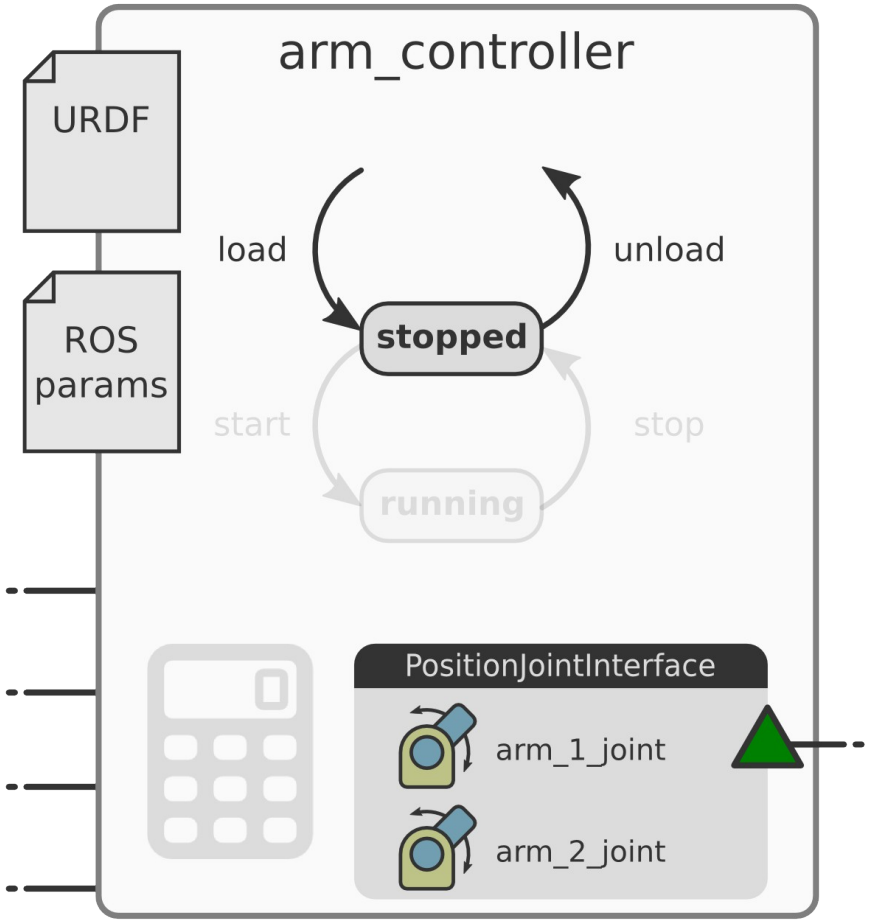
*not the same as resource **conflict** handling



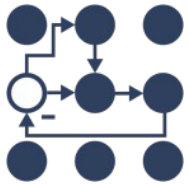
Controllers

Non real-time operations

- **load**
  - load + initialize plugin
  - check requisites (can fail)
    - hardware resource **existence***
    - configuration
  - setup ROS interfaces
- **unload**
  - destroy + unload plugin



*not the same as resource **conflict** handling

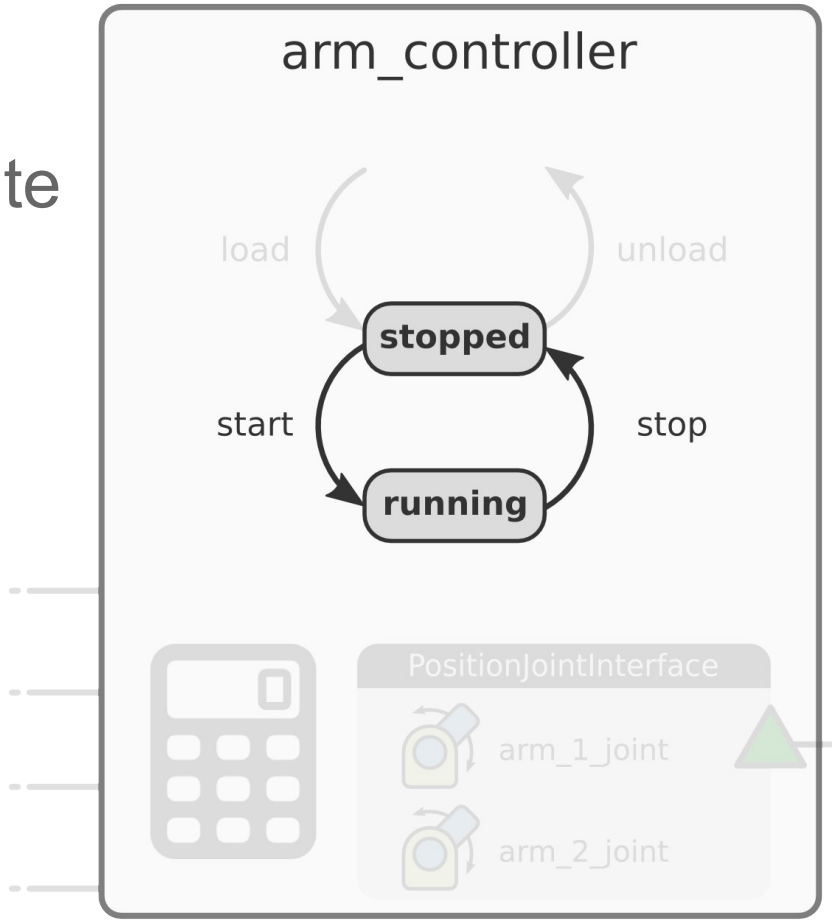


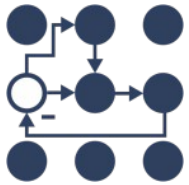
Controllers

Real-time safe operations

- **start** executed before first update
  - resource **conflict** handling
  - typical policy: semantic zero

- **stop** executed after last update
  - typical policy: cancel goals

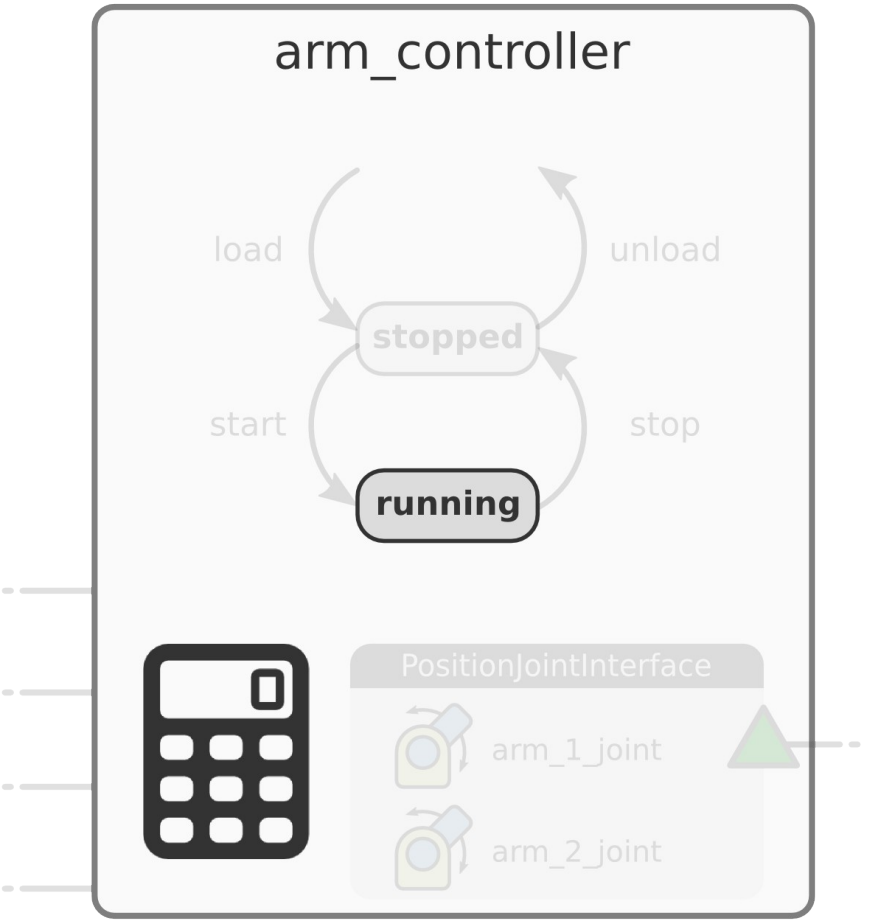




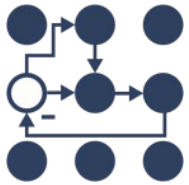
Controllers

Real-time safe* operations

- **update**
  - real-time safe computation
  - executed periodically



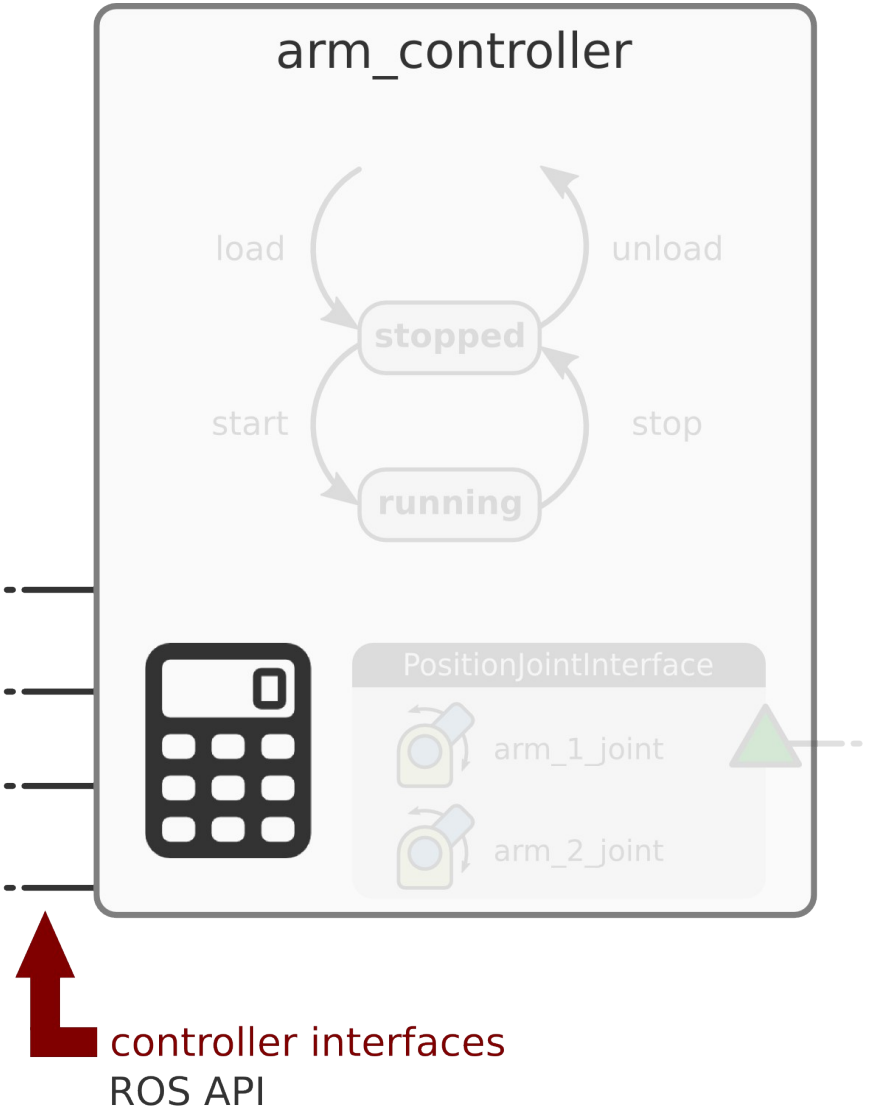
*requirement on implementation

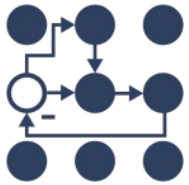


Controllers

Non real-time operations

- **callbacks**
  - non real-time computation
  - executed asynchronously

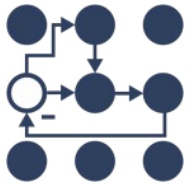




Controllers

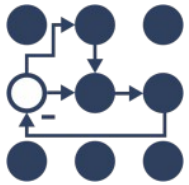
Summary

- Dynamically loadable **plugins**
- Interface defines a simple **state machine**
- Interface clearly **separates**
  - operations that are **non real-time**
  - operations required to be **real-time safe**
- Computation
  - **controller update:** periodic, real-time safe
  - **ROS API callbacks:** asynchronous, non real-time



control_toolbox

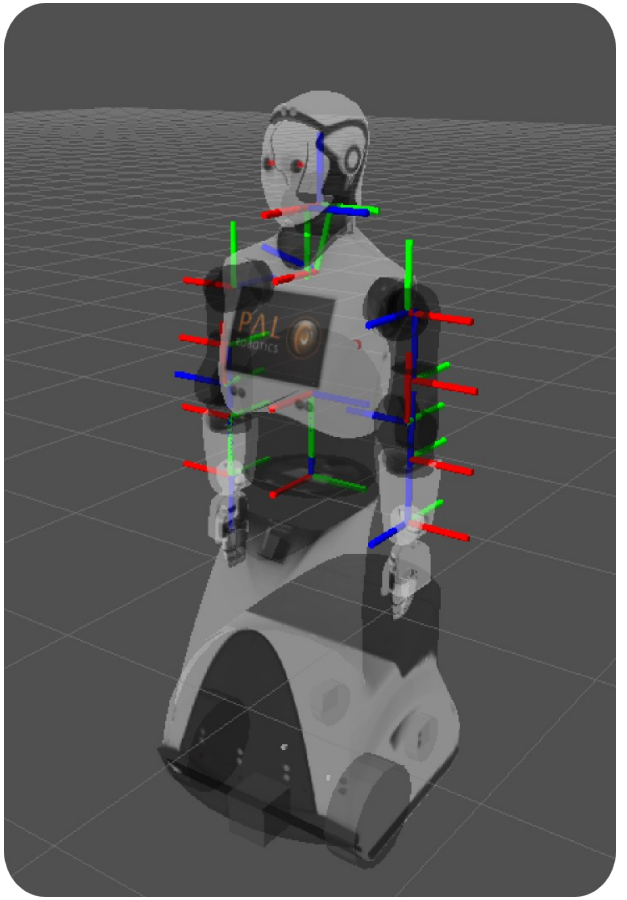
- Tools useful for writing **controllers** or **robot abstractions**
  - **Pid** PID loop
  - **SineSweep** for joint frequency analysis
  - **Dither** white noise generator
  - **LimitedProxy** for convergence without overshoot
  - ...

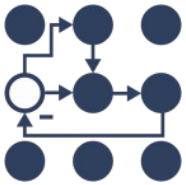


ros_controllers repo

Sensor state reporting

- **joint_state_controller**
  - publishes: **sensor_msgs/JointState** topic
- **imu_sensor_controller**
  - publishes: **sensor_msgs/Imu** topics
- **force_torque_sensor_controller**
  - publishes: **geometry_msgs/Wrench** topics

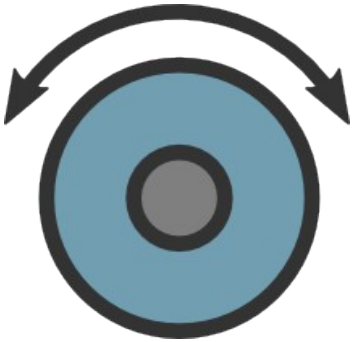


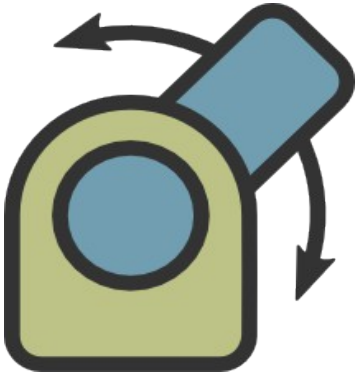


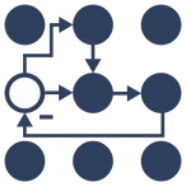
ros_controllers repo

General purpose

- **[position,velocity,effort]_controllers**
  - single-joint controllers in different control spaces



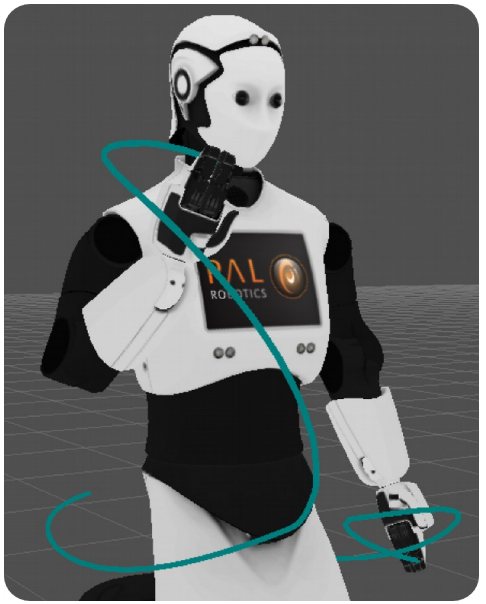


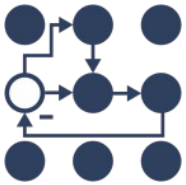


ros_controllers repo

joint_trajectory_controller

- multi-joint trajectory interpolator
- commands:
  - **control_msgs/FollowJointTrajectory** action
  - **trajectory_msgs/JointTrajectory** topic
- compatible with: **MoveIt!**

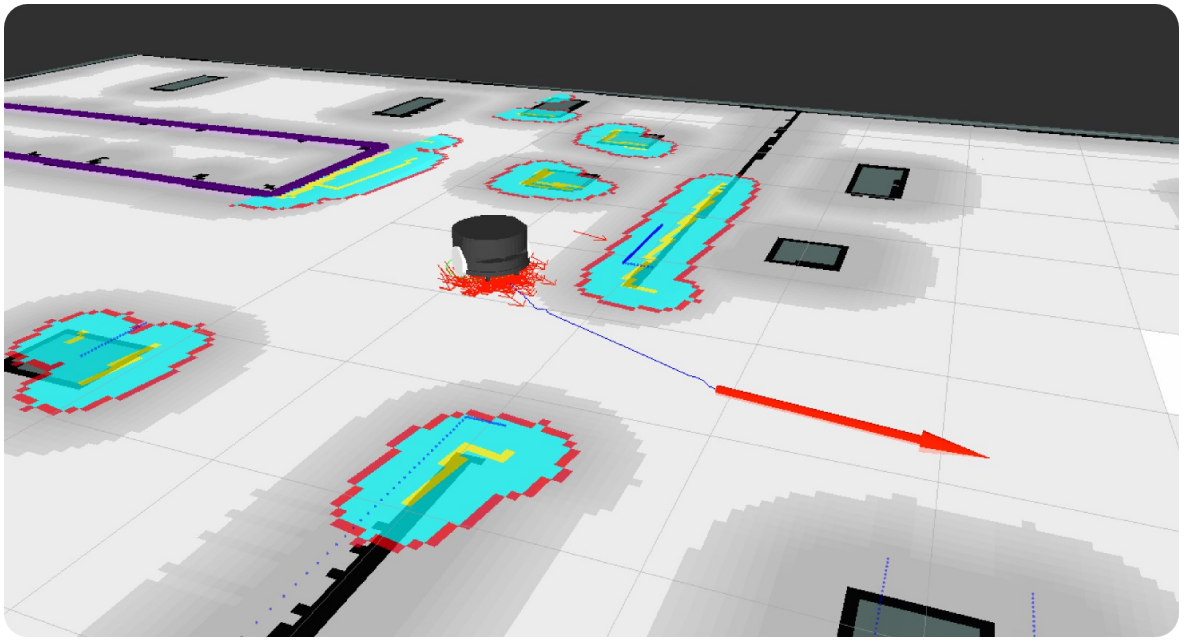


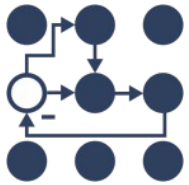


ros_controllers repo

Navigation

- **diff_drive_controller**
  - commands: **geometry_msgs/Twist** topic
  - publishes: odometry to **tf** and **nav_msgs/Odometry** topic
  - compatible with: the **ROS navigation stack**



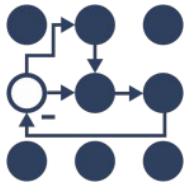


ros_controllers repo

Manipulation

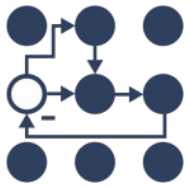
- **gripper_action_controller**
  - single-dof gripper controller
  - commands:
    - **control_msgs::GripperCommandAction** action
  - compatible with: **MoveIt!**

source: Willow Garage Flickr photostream

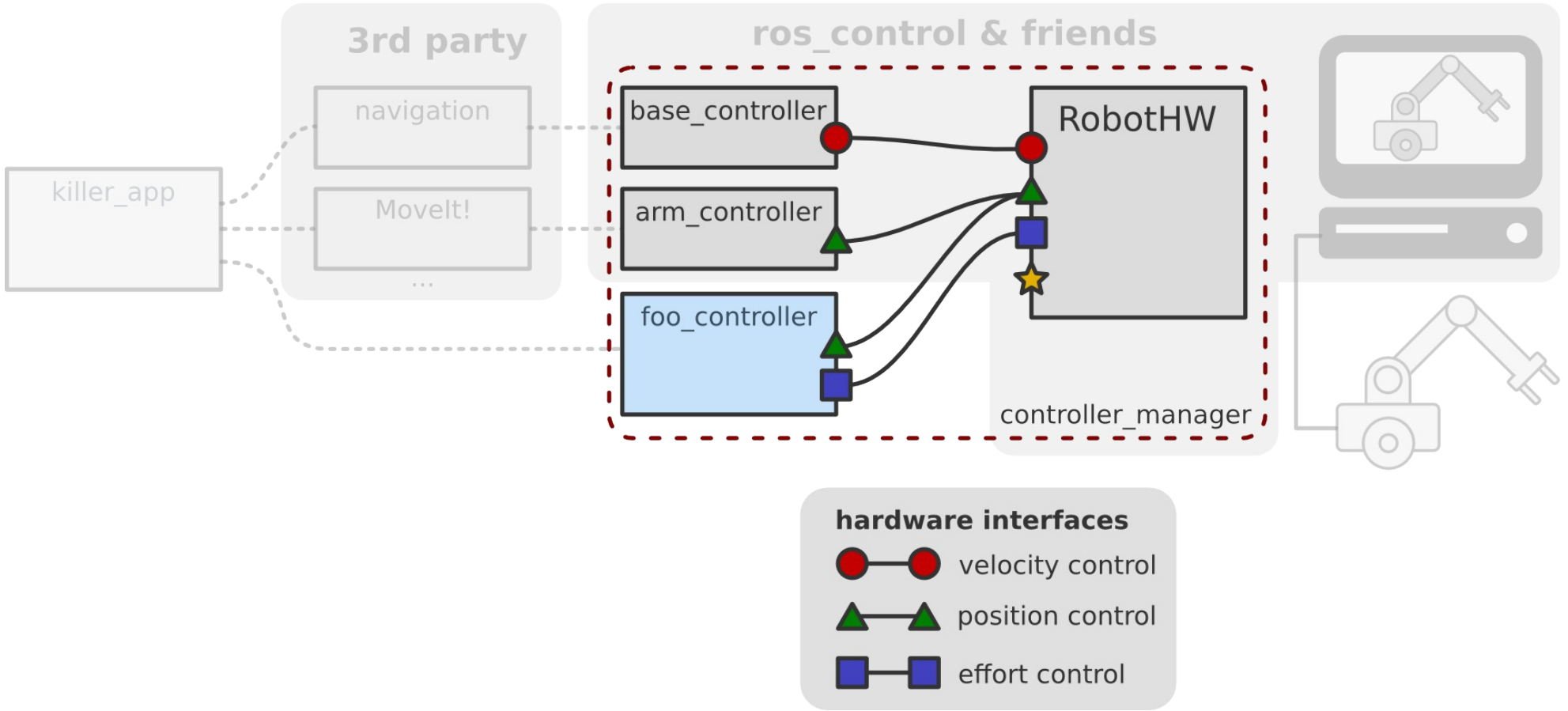


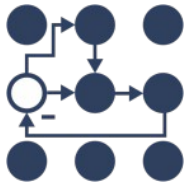
ROS control – an overview

- Big picture and goals
- ROS control & friends
  - Setting up a robot
  - Controllers
  - **The control loop**
- Demo
- Robots using ROS control



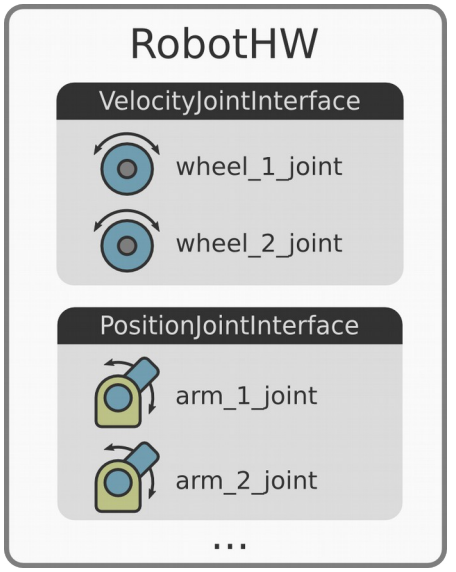
The control loop



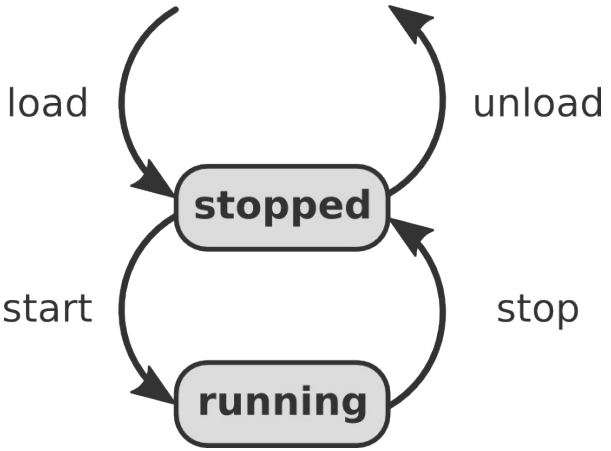


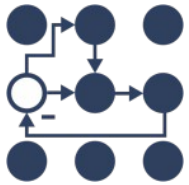
controller_manager

- Robot resource management
  - knows available resources
  - enforces resource conflict policy



- Controller lifecycle management
  - **transitions** controller state machine
  - **updates** running controllers
  - **periodic**, **serialized** updates

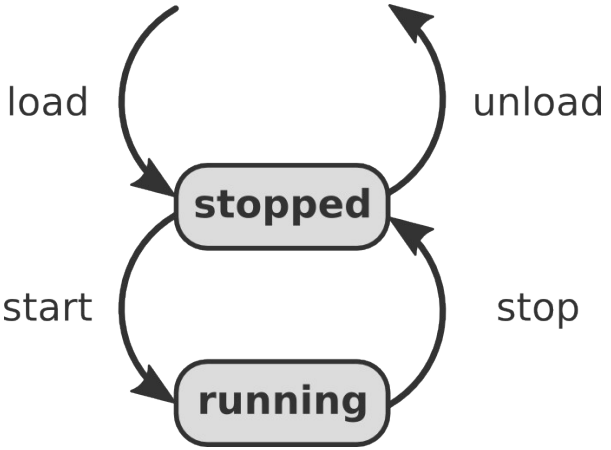


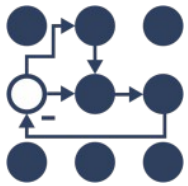


controller_manager

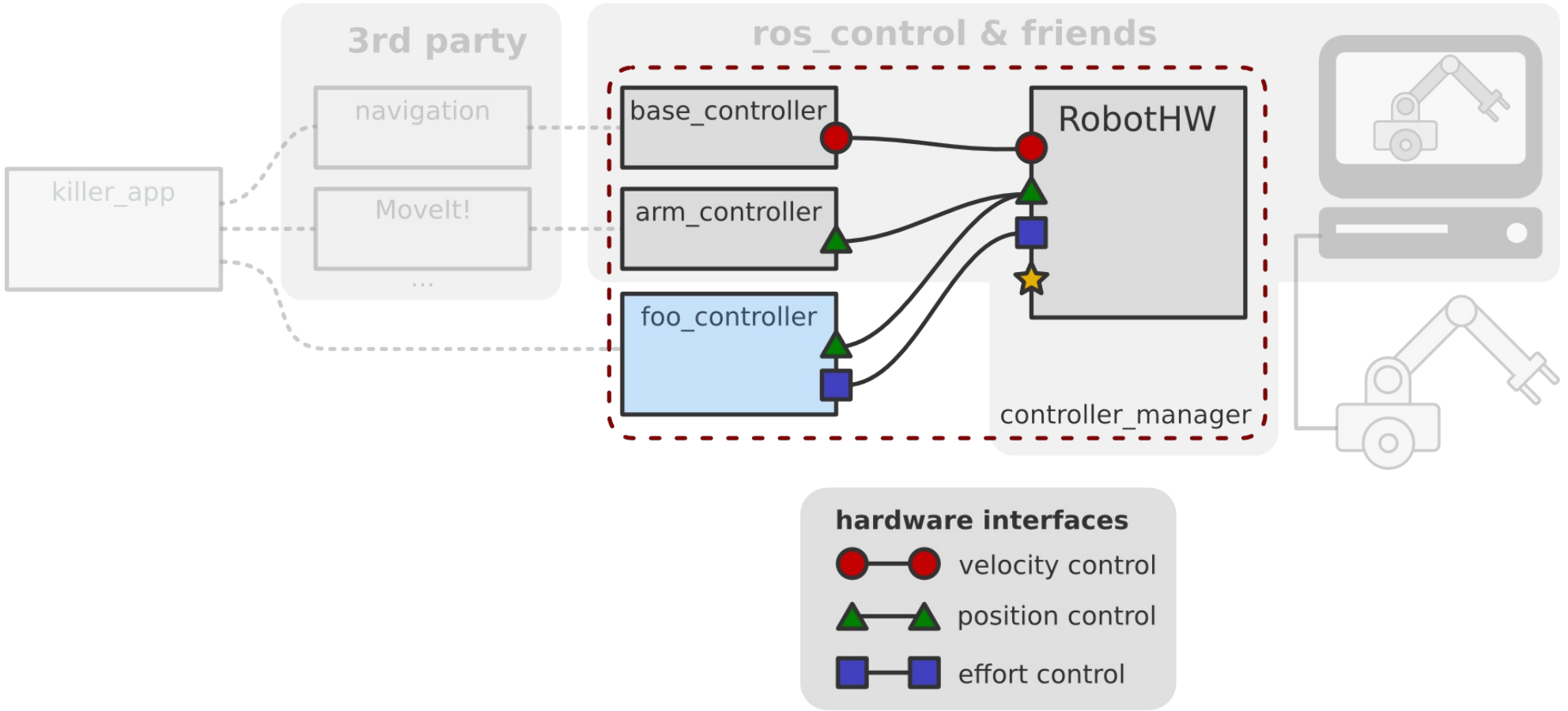
ROS service API

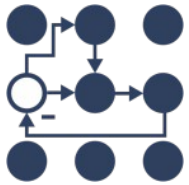
- Controller lifecycle management
  - load_controller
  - unload_controller
  - switch_controller
- Queries
  - list_controllers
  - list_controller_types
- Other
  - reload_controller_libraries



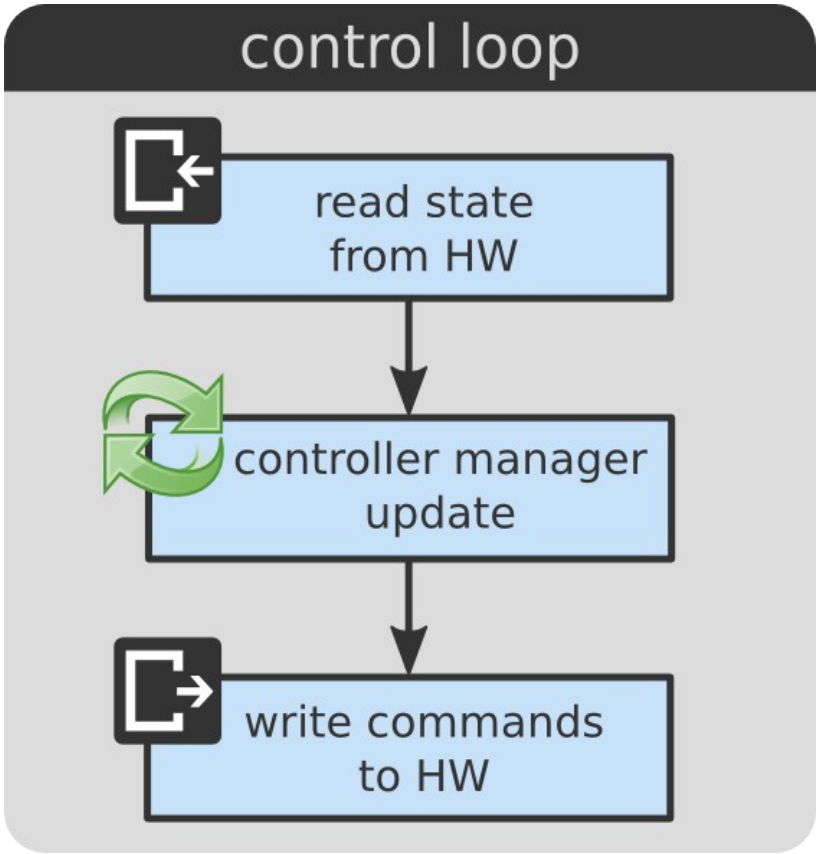


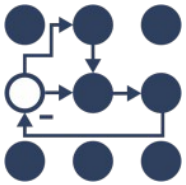
The control loop





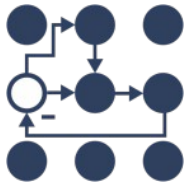
The control loop





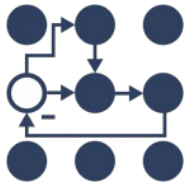
The control loop

```
#include <ros/ros.h>
#include <my_robot/my_robot.h>
#include <controller_manager/controller_manager.h>

int main(int argc, char **argv)
{
  // Setup
  ros::init(argc, argv, "my_robot");

  MyRobot::MyRobot robot;
  controller_manager::ControllerManager cm(&robot);

  ros::AsyncSpinner spinner(1);
  spinner.start();

  // Control loop
  ros::Time prev_time = ros::Time::now();
  ros::Rate rate(10.0);

  while (ros::ok())
  {
     const ros::Time     time   = ros::Time::now();
     const ros::Duration period = time - prev_time;

     robot.read();
     cm.update(time, period);
     robot.write();

     rate.sleep();
  }
  return 0;
}
```



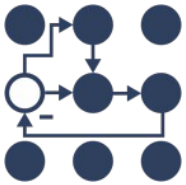
The control loop

```
#include <ros/ros.h>
#include <my_robot/my_robot.h>
#include <controller_manager/controller_manager.h>

int main(int argc, char **argv)
{
  // Setup
  ros::init(argc, argv, "my_robot");

  MyRobot::MyRobot robot;
  controller_manager::ControllerManager cm(&robot);

  ros::AsyncSpinner spinner(1);
  spinner.start();

  // Control loop
  ros::Time prev_time = ros::Time::now();
  ros::Rate rate(10.0);

  while (ros::ok())
  {
     const ros::Time     time   = ros::Time::now();
     const ros::Duration period = time - prev_time;

     robot.read();
     cm.update(time, period);
     robot.write();

     rate.sleep();
  }
  return 0;
}
```



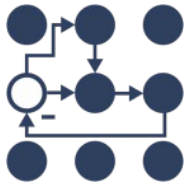
The control loop

```
#include <ros/ros.h>
#include <my_robot/my_robot.h>
#include <controller_manager/controller_manager.h>

int main(int argc, char **argv)
{
  // Setup
  ros::init(argc, argv, "my_robot");

  MyRobot::MyRobot robot;
  controller_manager::ControllerManager cm(&robot);

  ros::AsyncSpinner spinner(1);
  spinner.start();

  // Control loop
  ros::Time prev_time = ros::Time::now();
  ros::Rate rate(10.0);

  while (ros::ok())
  {
     const ros::Time     time   = ros::Time::now();
     const ros::Duration period = time - prev_time;

     robot.read();
     cm.update(time, period);
     robot.write();

     rate.sleep();
  }
  return 0;
}
```



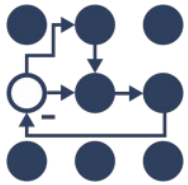
The control loop

```
#include <ros/ros.h>
#include <my_robot/my_robot.h>
#include <controller_manager/controller_manager.h>

int main(int argc, char **argv)
{
  // Setup
  ros::init(argc, argv, "my_robot");

  MyRobot::MyRobot robot;
  controller_manager::ControllerManager cm(&robot);

  ros::AsyncSpinner spinner(1);
  spinner.start();

  // Control loop
  ros::Time prev_time = ros::Time::now();
  ros::Rate rate(10.0);

  while (ros::ok())
  {
     const ros::Time     time   = ros::Time::now();
     const ros::Duration period = time - prev_time;

     robot.read();
     cm.update(time, period);
     robot.write();

     rate.sleep();
  }
  return 0;
}
```



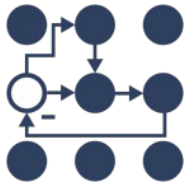
The control loop

```
#include <ros/ros.h>
#include <my_robot/my_robot.h>
#include <controller_manager/controller_manager.h>

int main(int argc, char **argv)
{
  // Setup
  ros::init(argc, argv, "my_robot");

  MyRobot::MyRobot robot;
  controller_manager::ControllerManager cm(&robot);

  ros::AsyncSpinner spinner(1);
  spinner.start();

  // Control loop
  ros::Time prev_time = ros::Time::now();
  ros::Rate rate(10.0);

  while (ros::ok())
  {
     const ros::Time     time   = ros::Time::now();
     const ros::Duration period = time - prev_time;

     robot.read();
     cm.update(time, period);
     robot.write();

     rate.sleep();
  }
  return 0;
}
```



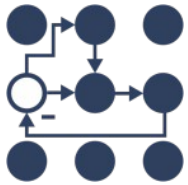
The control loop

```
#include <ros/ros.h>
#include <my_robot/my_robot.h>
#include <controller_manager/controller_manager.h>

int main(int argc, char **argv)
{
  // Setup
  ros::init(argc, argv, "my_robot");

  MyRobot::MyRobot robot;
  controller_manager::ControllerManager cm(&robot);

  ros::AsyncSpinner spinner(1);
  spinner.start();

  // Control loop
  ros::Time prev_time = ros::Time::now();
  ros::Rate rate(10.0);

  while (ros::ok())
  {
     const ros::Time     time   = ros::Time::now();
     const ros::Duration period = time - prev_time;

     robot.read();
     cm.update(time, period);
     robot.write();

     rate.sleep();
  }
  return 0;
}
```



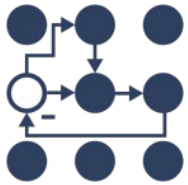
The control loop

```
#include <ros/ros.h>
#include <my_robot/my_robot.h>
#include <controller_manager/controller_manager.h>

int main(int argc, char **argv)
{
  // Setup
  ros::init(argc, argv, "my_robot");

  MyRobot::MyRobot robot;
  controller_manager::ControllerManager cm(&robot);

  ros::AsyncSpinner spinner(1);
  spinner.start();

  // Control loop
  ros::Time prev_time = ros::Time::now();
  ros::Rate rate(10.0);

  while (ros::ok())
  {
    const ros::Time     time   = ros::Time::now();
    const ros::Duration period = time - prev_time;

    robot.read();
    cm.update(time, period);
    robot.write();

    rate.sleep();
  }
  return 0;
}
```



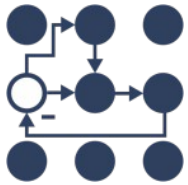
The control loop

```
#include <ros/ros.h>
#include <my_robot/my_robot.h>
#include <controller_manager/controller_manager.h>

int main(int argc, char **argv)
{
  // Setup
  ros::init(argc, argv, "my_robot");

  MyRobot::MyRobot robot;
  controller_manager::ControllerManager cm(&robot);

  ros::AsyncSpinner spinner(1);
  spinner.start();

  // Control loop
  ros::Time prev_time = ros::Time::now();
  ros::Rate rate(10.0);

  while (ros::ok())
  {
     const ros::Time     time   = ros::Time::now();
     const ros::Duration period = time - prev_time;

     robot.read();
     cm.update(time, period);
     robot.write();

     rate.sleep();
  }
  return 0;
}
```



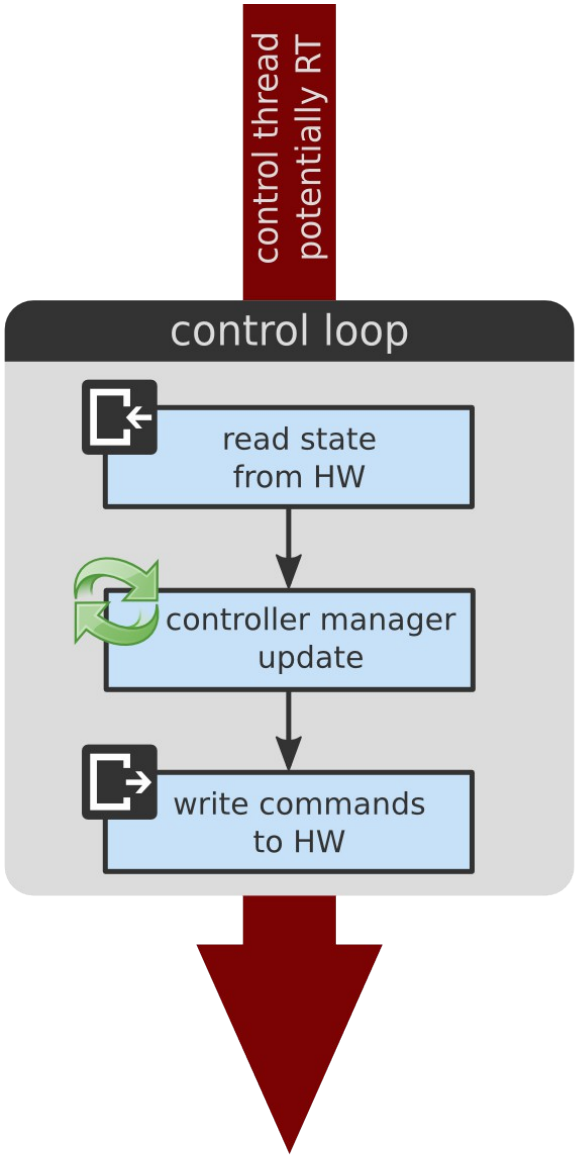
The control loop

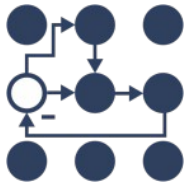
```
#include <ros/ros.h>
#include <my_robot/my_robot.h>
#include <controller_manager/controller_manager.h>

int main(int argc, char **argv)
{
  // Setup
  ros::init(argc, argv, "my_robot");

  MyRobot::MyRobot robot;
  controller_manager::ControllerManager cm(&robot);

  ros::AsyncSpinner spinner(1);
  spinner.start();

  // Control loop
  ros::Time prev_time = ros::Time::now();
  ros::Rate rate(10.0);

  while (ros::ok())
  {
     const ros::Time     time   = ros::Time::now();
     const ros::Duration period = time - prev_time;

     robot.read();
     cm.update(time, period);
     robot.write();

     rate.sleep();
  }
  return 0;
}
```



The control loop

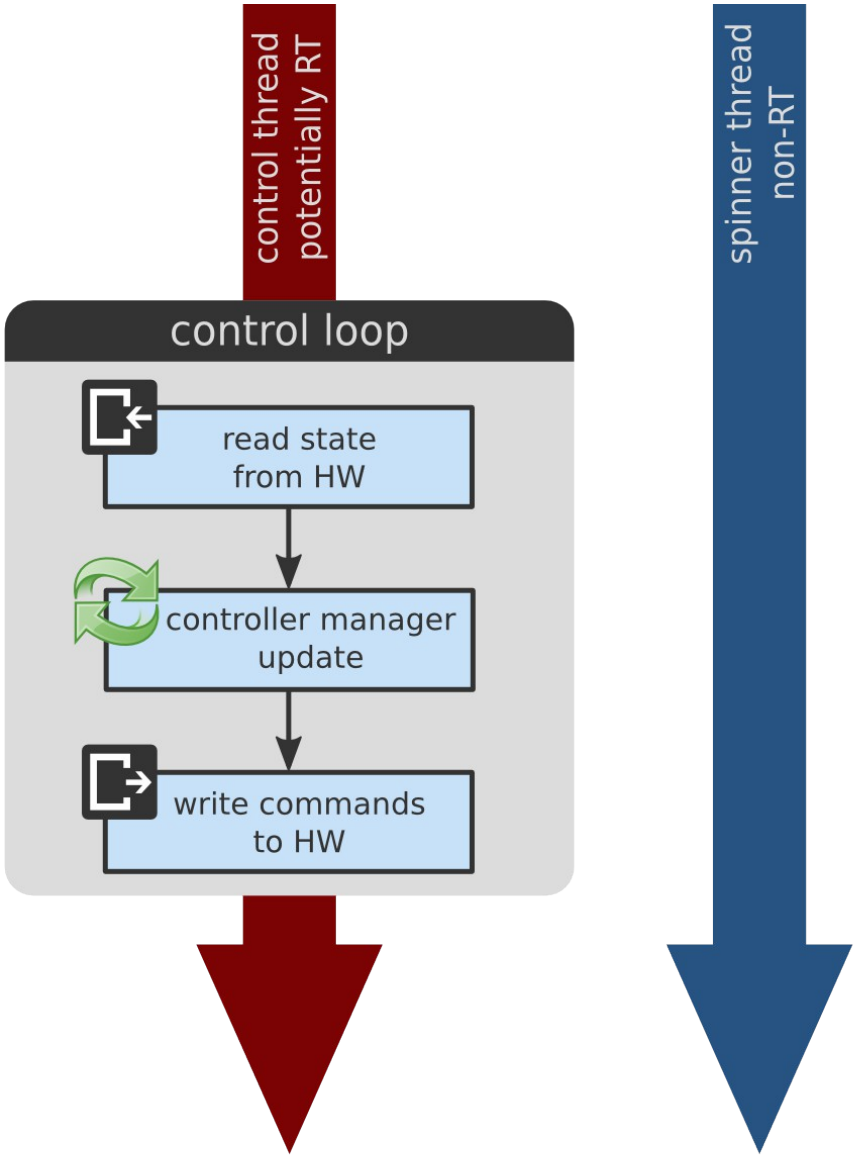
```
#include <ros/ros.h>
#include <my_robot/my_robot.h>
#include <controller_manager/controller_manager.h>

int main(int argc, char **argv)
{
  // Setup
  ros::init(argc, argv, "my_robot");

  MyRobot::MyRobot robot;
  controller_manager::ControllerManager cm(&robot);

  ros::AsyncSpinner spinner(1);
  spinner.start();

  // Control loop
  ros::Time prev_time = ros::Time::now();
  ros::Rate rate(10.0);

  while (ros::ok())
  {
    const ros::Time     time   = ros::Time::now();
    const ros::Duration period = time - prev_time;

    robot.read();
    cm.update(time, period);
    robot.write();

    rate.sleep();
  }
  return 0;
}
```

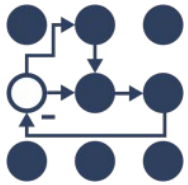




The control loop

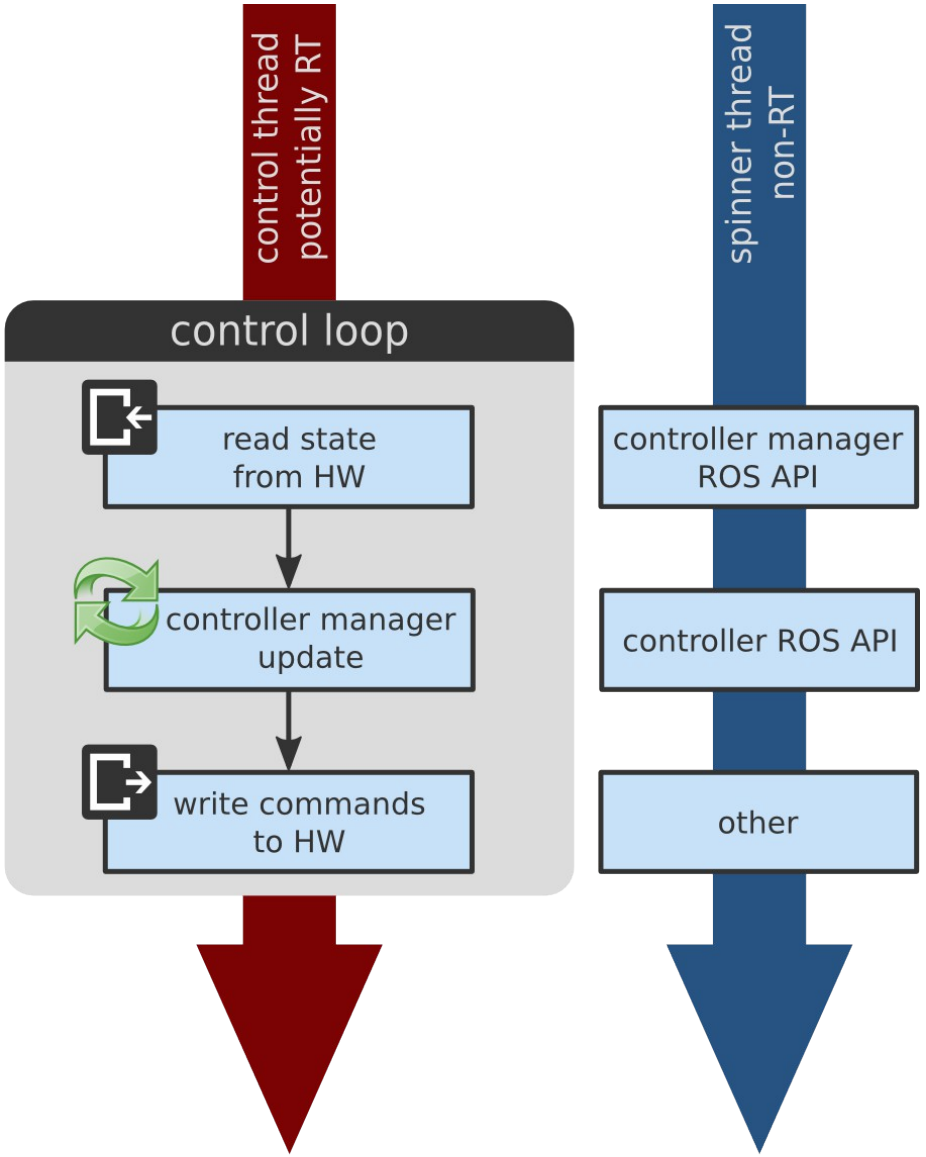
```
#include <ros/ros.h>
#include <my_robot/my_robot.h>
#include <controller_manager/controller_manager.h>

int main(int argc, char **argv)
{
  // Setup
  ros::init(argc, argv, "my_robot");

  MyRobot::MyRobot robot;
  controller_manager::ControllerManager cm(&robot);

  ros::AsyncSpinner spinner(1);
  spinner.start();

  // Control loop
  ros::Time prev_time = ros::Time::now();
  ros::Rate rate(10.0);

  while (ros::ok())
  {
    const ros::Time     time   = ros::Time::now();
    const ros::Duration period = time - prev_time;

    robot.read();
    cm.update(time, period);
    robot.write();

    rate.sleep();
  }
  return 0;
}
```

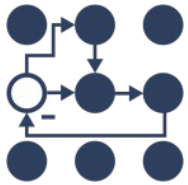




The control loop

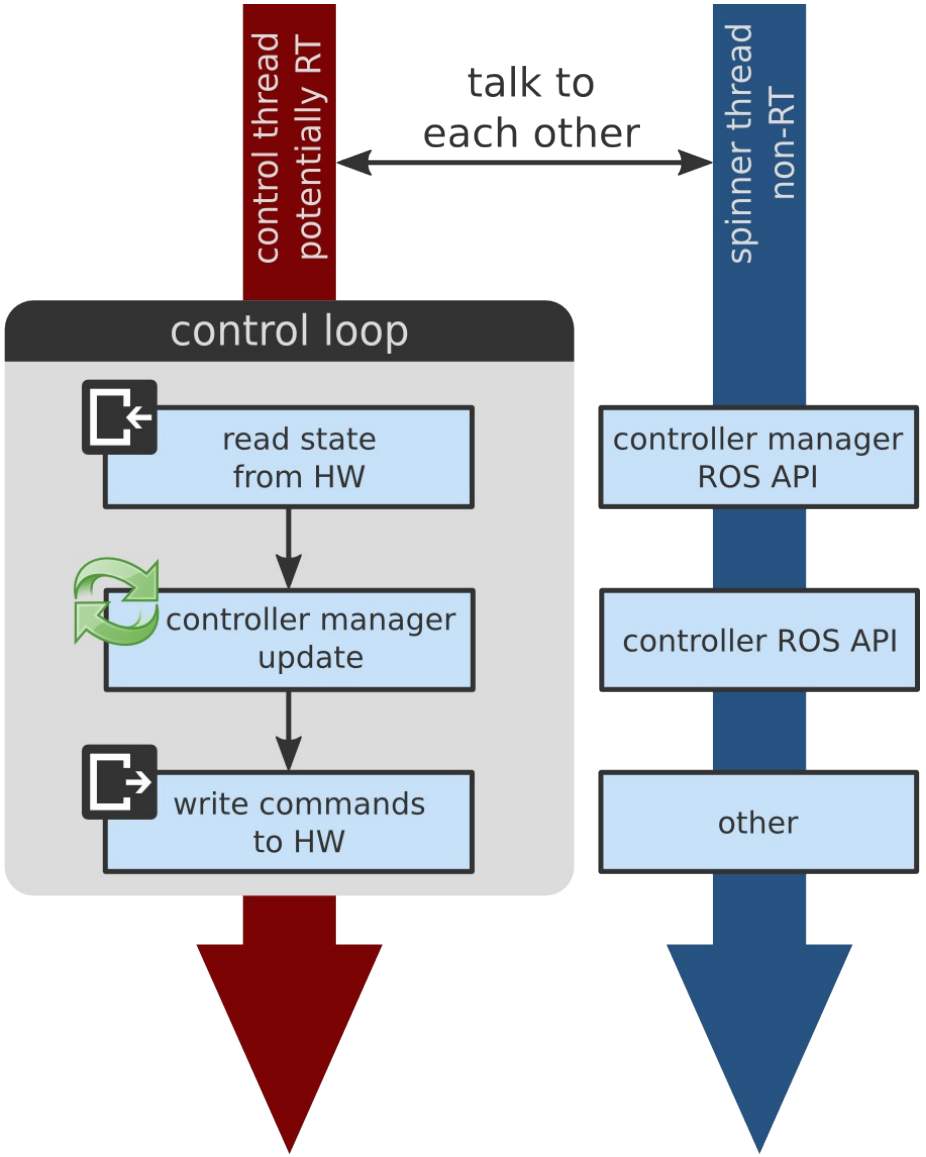
```
#include <ros/ros.h>
#include <my_robot/my_robot.h>
#include <controller_manager/controller_manager.h>

int main(int argc, char **argv)
{
  // Setup
  ros::init(argc, argv, "my_robot");

  MyRobot::MyRobot robot;
  controller_manager::ControllerManager cm(&robot);

  ros::AsyncSpinner spinner(1);
  spinner.start();

  // Control loop
  ros::Time prev_time = ros::Time::now();
  ros::Rate rate(10.0);

  while (ros::ok())
  {
    const ros::Time     time   = ros::Time::now();
    const ros::Duration period = time - prev_time;

    robot.read();
    cm.update(time, period);
    robot.write();

    rate.sleep();
  }
  return 0;
}
```

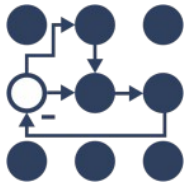




The control loop

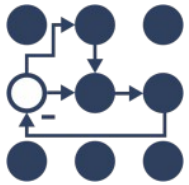
```
#include <ros/ros.h>
#include <my_robot/my_robot.h>
#include <controller_manager/controller_manager.h>

int main(int argc, char **argv)
{
  // Setup
  ros::init(argc, argv, "my_robot");

  MyRobot::MyRobot robot;
  controller_manager::ControllerManager cm(&robot);

  ros::AsyncSpinner spinner(1);
  spinner.start();

  // Control loop
  ros::Time prev_time = ros::Time::now();
  ros::Rate rate(10.0);

  while (ros::ok())
  {
    const ros::Time     time   = ros::Time::now();
    const ros::Duration period = time - prev_time;

    robot.read();
    cm.update(time, period);
    robot.write();

    rate.sleep();
  }
  return 0;
}
```



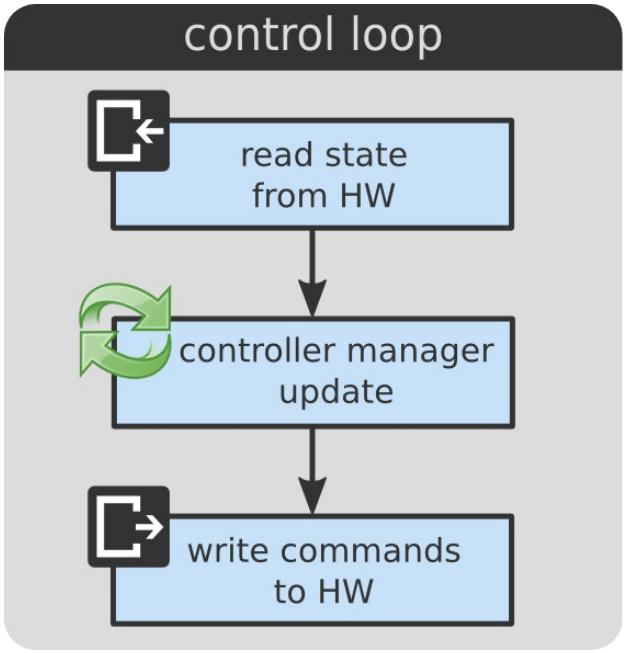


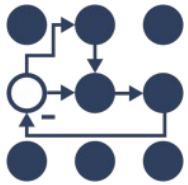
realtime_tools

- Tools usable from a **real-time** thread
  - **RealtimePublisher** Publish to a ROS topic
  - **RealtimeBuffer** Share resource with non-RT thread
  - **RealtimeClock** Query system clock
  - ...

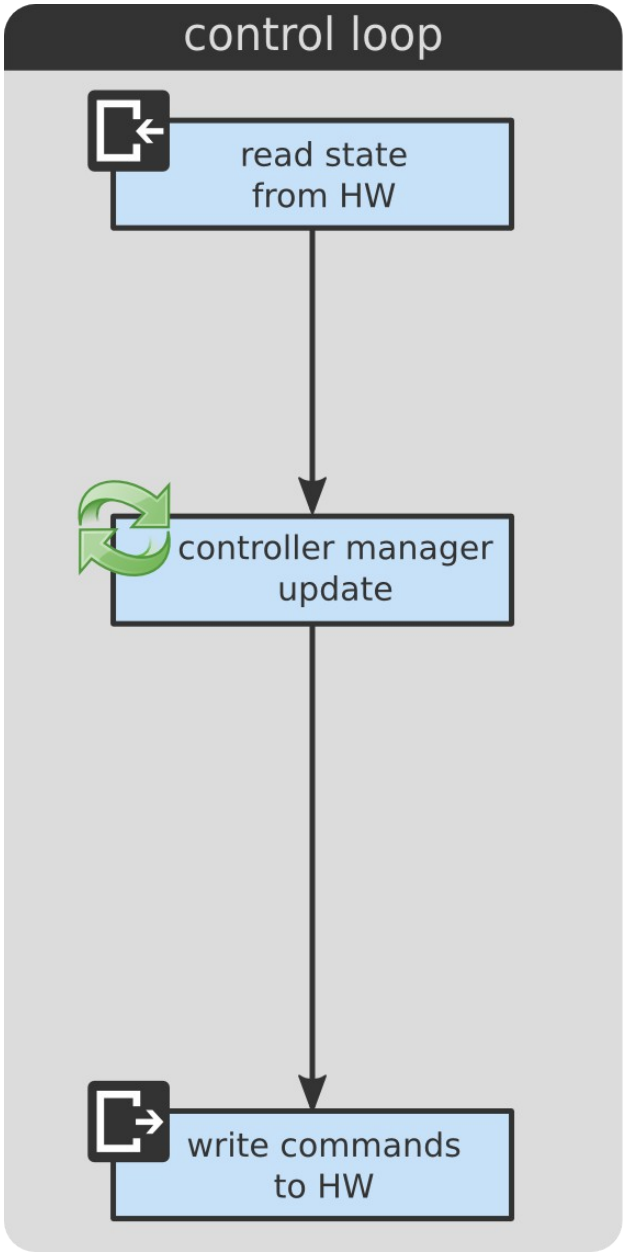


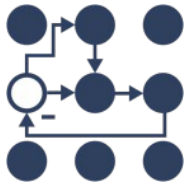
The control loop – Part II



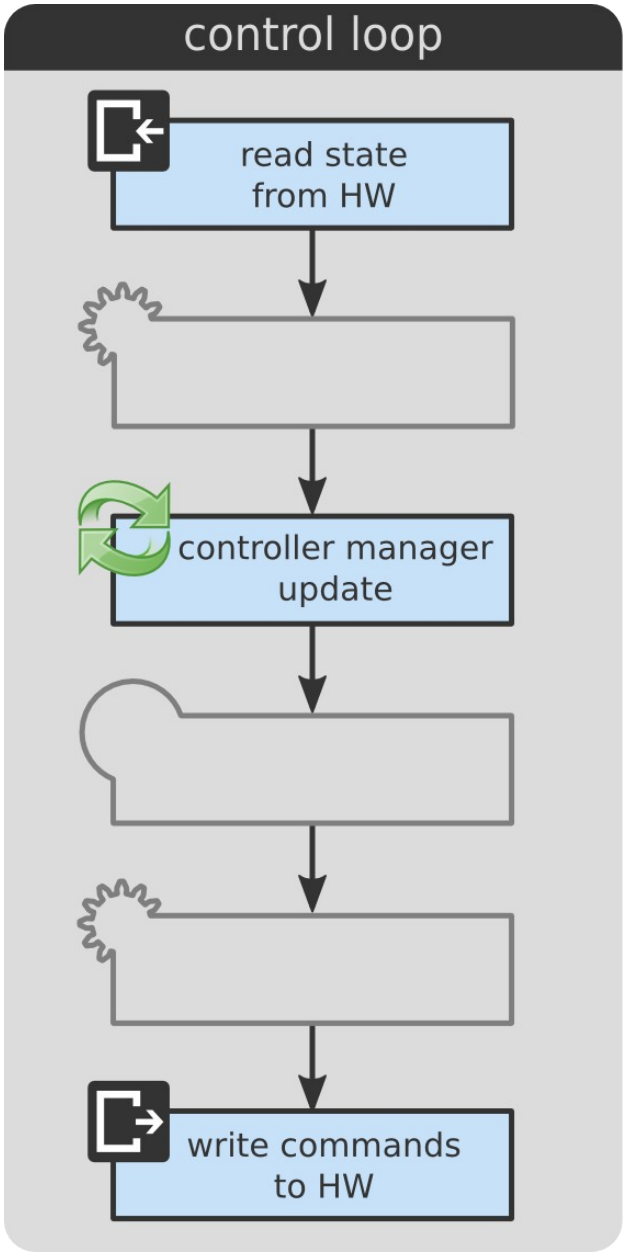


The control loop – Part II

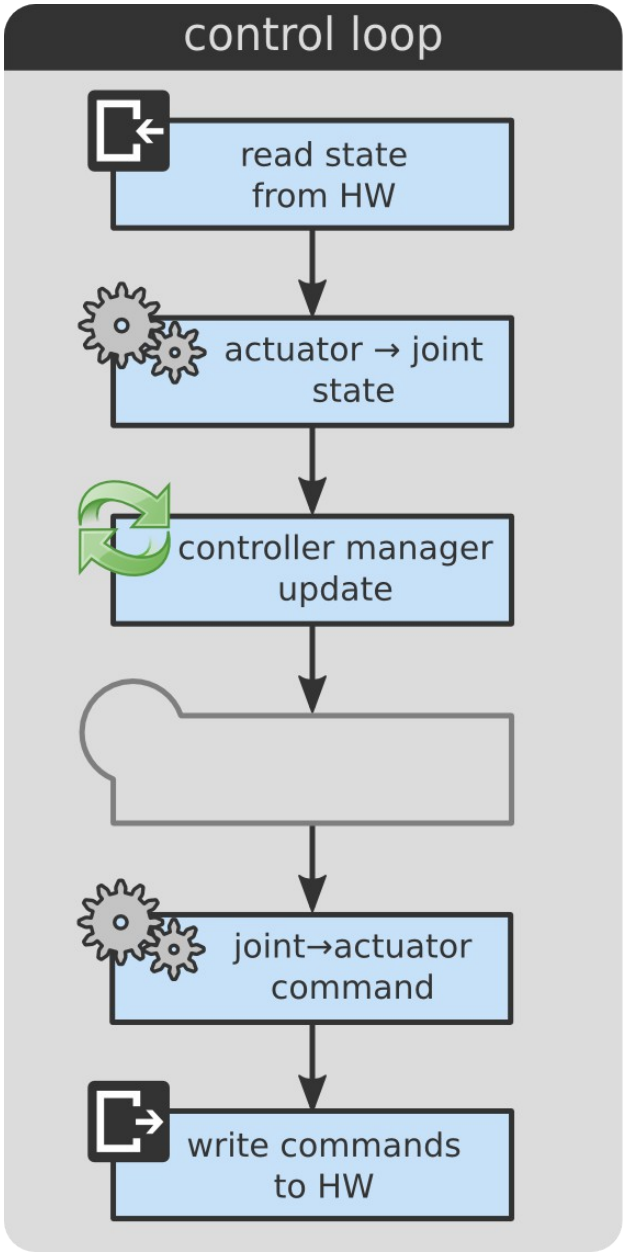




The control loop – Part II

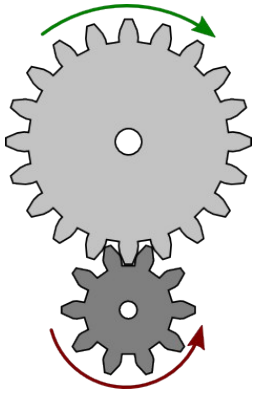


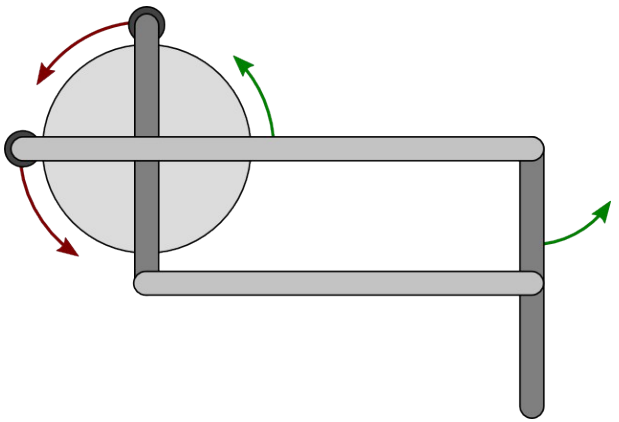
The control loop – Part II



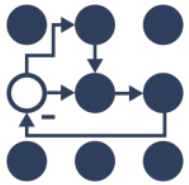
transmission_interface

- Mechanical transmission representation
  - propagate between **actuator** ↔ **joint** spaces...
  - **position**, **velocity** and **effort** variables
- Available transmissions
  - Simple reducer
  - Four-bar linkage
  - Differential



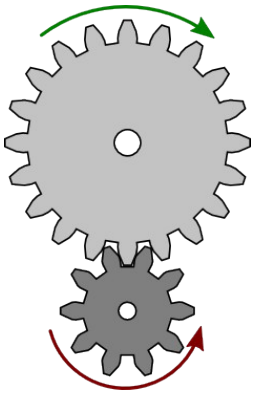


source: cgimotion.com



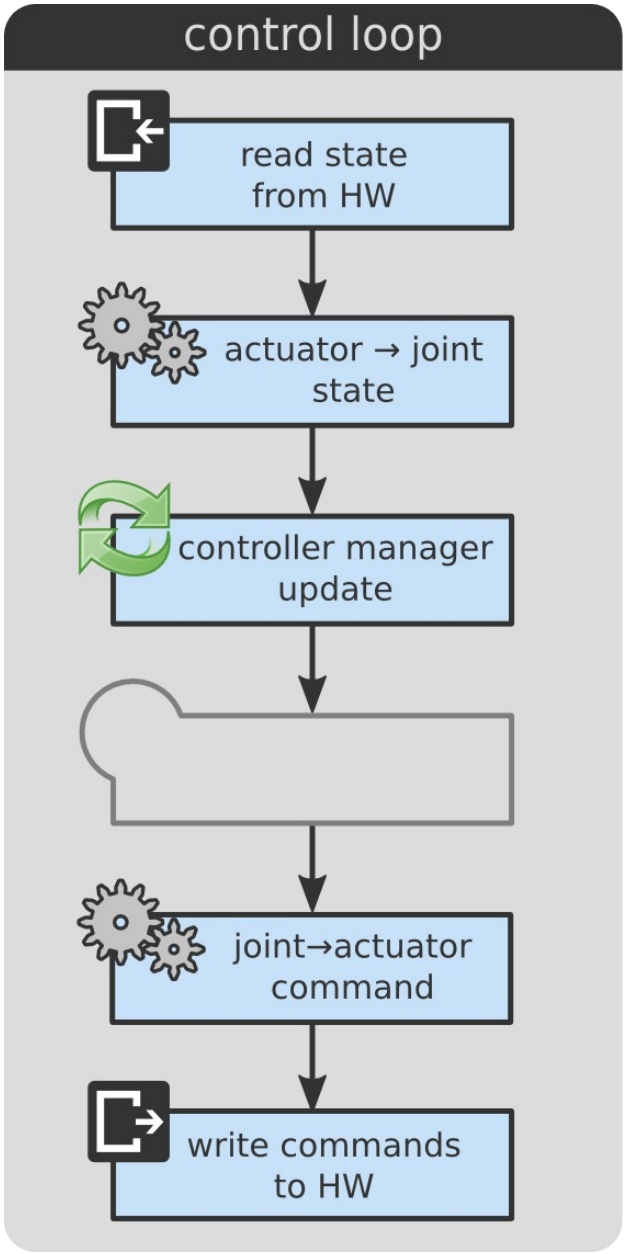
transmission_interface

- **Plugins** for loading from URDF
  - **Simplifies** populating **RobotHW** interfaces

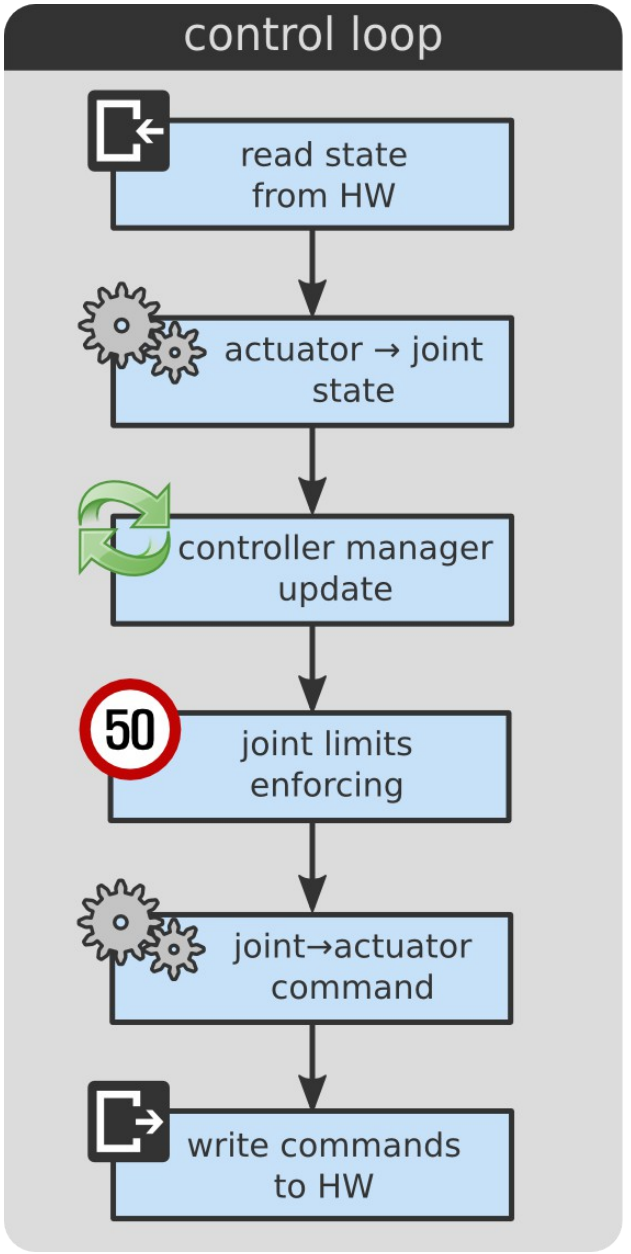


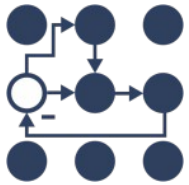
```
<transmission name="arm_1_trans">
  <type>transmission_interface/SimpleTransmission</type>
  <actuator name="arm_1_motor" >
    <mechanicalReduction>42</mechanicalReduction>
  </actuator>
  <joint name="arm_1_joint">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
</transmission>
```

The control loop – Part II



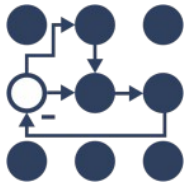
The control loop – Part II





joint_limits_interface

- **Last line of defense** before commanding HW
  - Don't trust controllers ;-)
- **Simple**, **fast**, **low-level** strategies

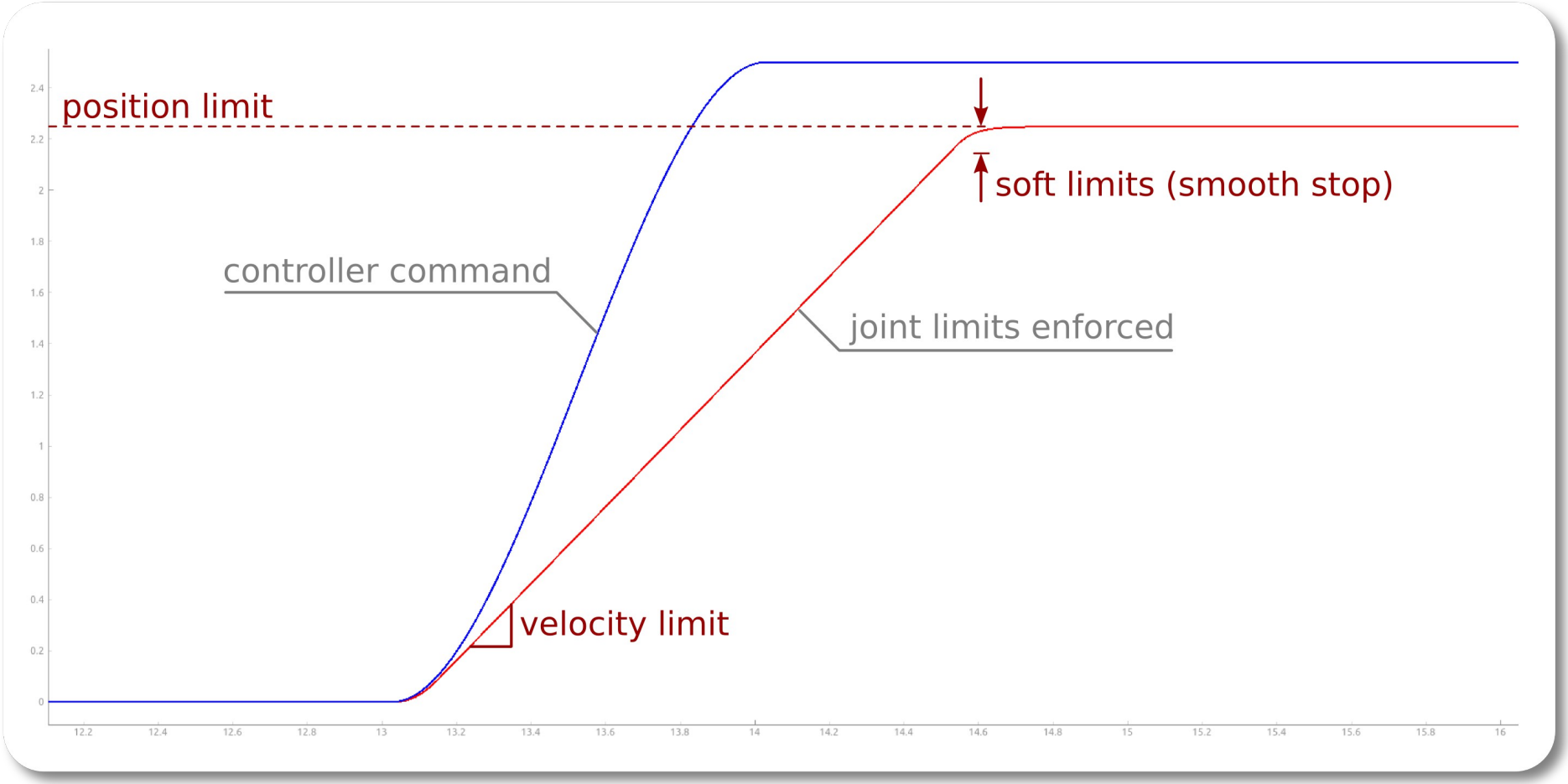


joint_limits_interface

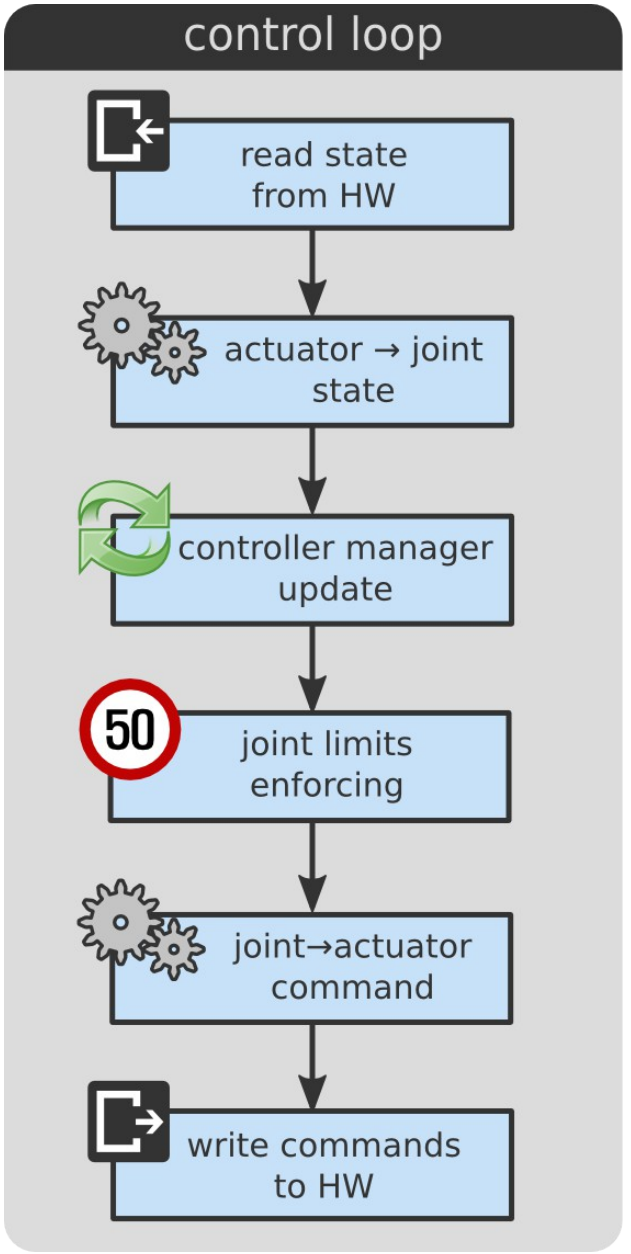
- Data structures for **representing** joint limits
- Methods for **populating** them
  - URDF
  - yaml (à la MoveIt! + jerk)
- Methods for **enforcing** joint limits
  - soft limits (PR2-like) and clamping
  - for different hardware interfaces

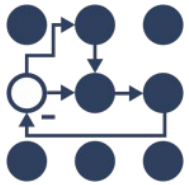
joint_limits_interface

Example

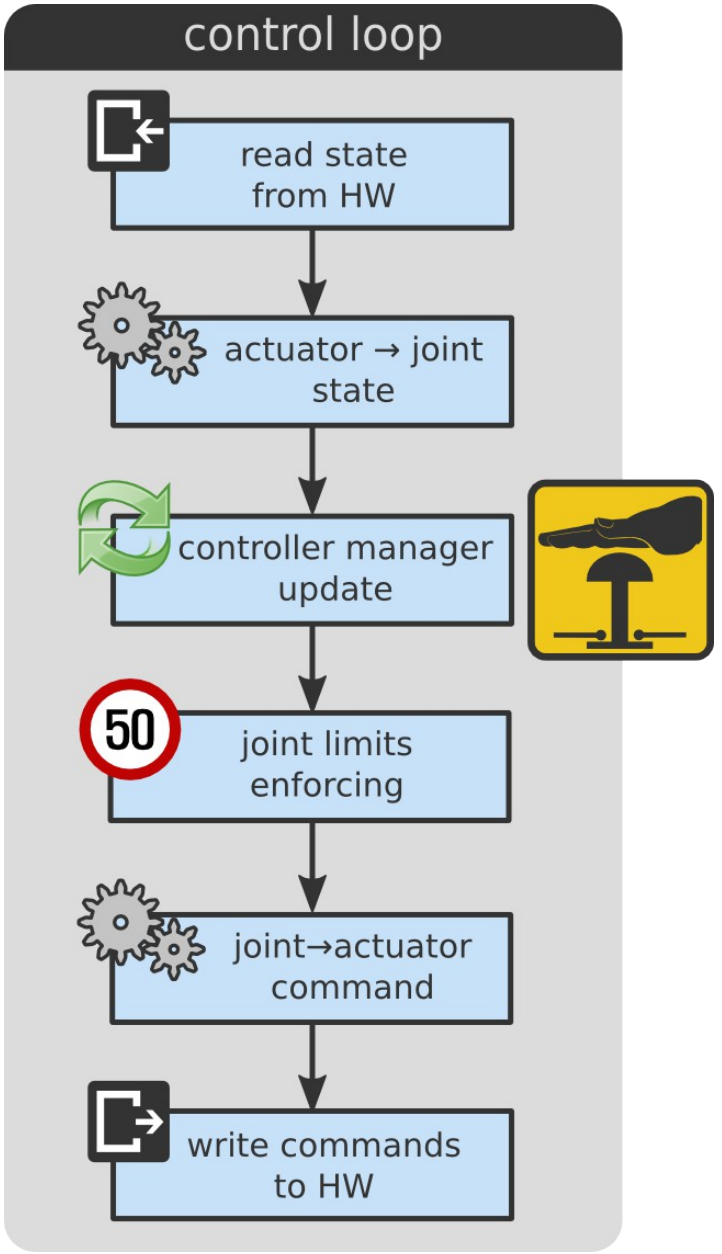


The control loop – Part II





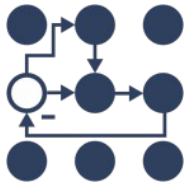
The control loop – Part II



E-stop handling

- Typically **robot-specific**
  - **Emergency** vs **protective** (risk reducing) stop
  - Remove **energy sources** vs **hazard control**
  - ...

Relevant standards:
ISO 10218-1:2011 Robots and robotic devices – Safety requirements for industrial robots
ISO 13482:2014 Robots and robotic devices – Safety requirements for personal care robots



E-stop handling

- **Recovery** behavior (release stop)
  - What's the **sane thing** to do?
  - Let **running controllers** decide!
- **Reset** controllers
  - on **controller_manager** update
  - **stop** + **start** + **update**, in a **single** control cycle
  - Typical policy:

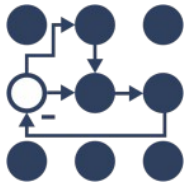
stop
cancel goals

+

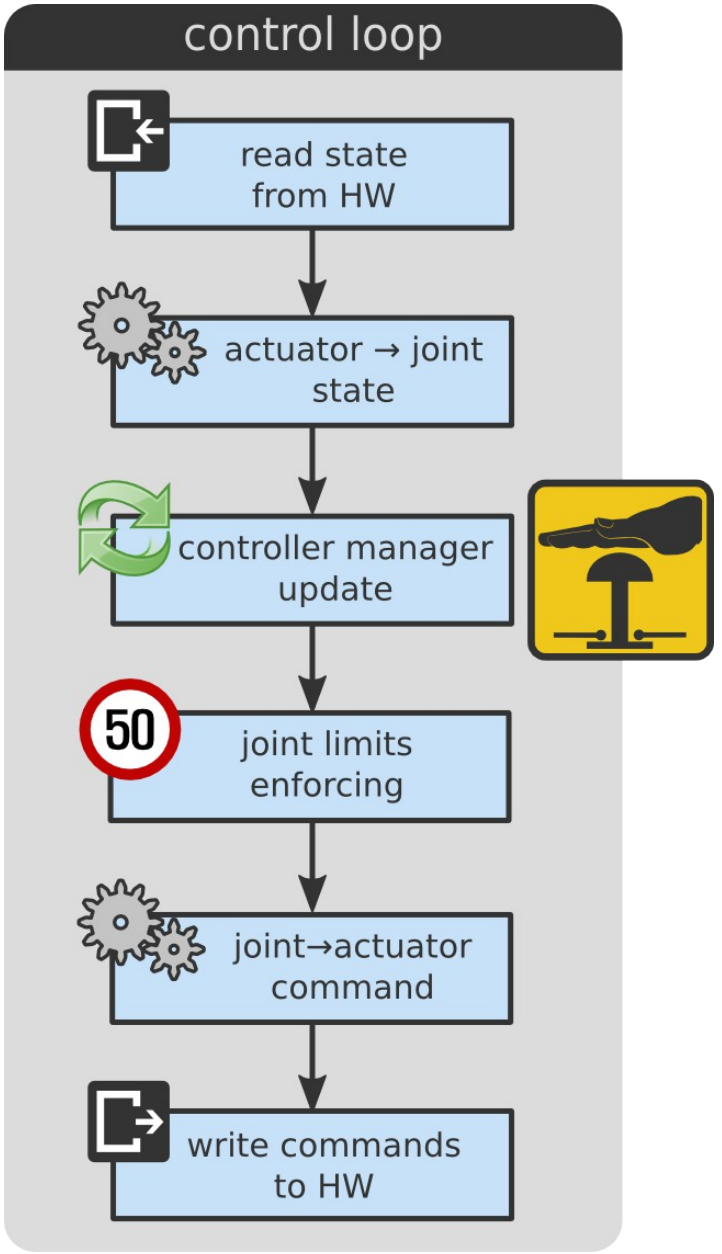
start
semantic zero

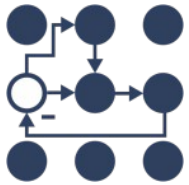
+

update
sane behavior!



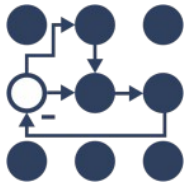
The control loop – Part II





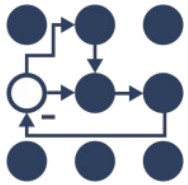
ROS control – an overview

- Big picture and goals
- ROS control & friends
  - Setting up a robot
  - Controllers
  - The control loop
- Demo
- Robots using ROS control



ROS control – an overview

- Big picture and goals
- ROS control & friends
  - Setting up a robot
  - Controllers
  - The control loop
- Demo
- Robots using ROS control



Robots using ROS control

Bauman Moscow State University

Space robotics based on functional model testbed

Russia

- Kawasaki FS 020N
- Kawasaki FS 003N
- Schunk EZN64 gripper
- Schunk SDH hand
- ATI delta F/T sensor

ros_control + ROS-I

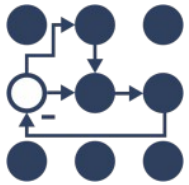
Robots using ROS control

Fraunhofer IPA

Robots and assistive systems – Germany

- Schunk LWA4d
  7 dof manipulator

- Schunk LWA4p
  6dof manipulator

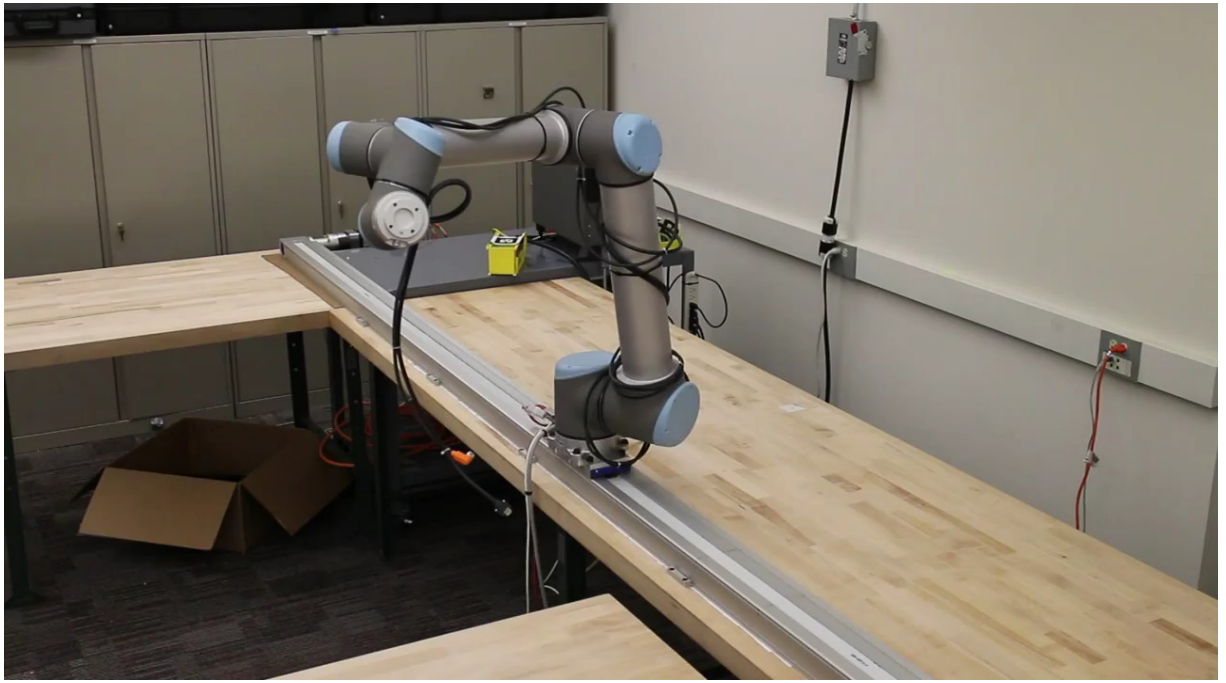


Robots using ROS control

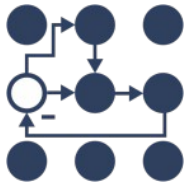
Georgia Tech (IRIM)

Collaborative assembly – USA

- UR-10
  6dof manipulator
- Indradyn motor
  In Schunk linear rail



ros_control + ROS-I

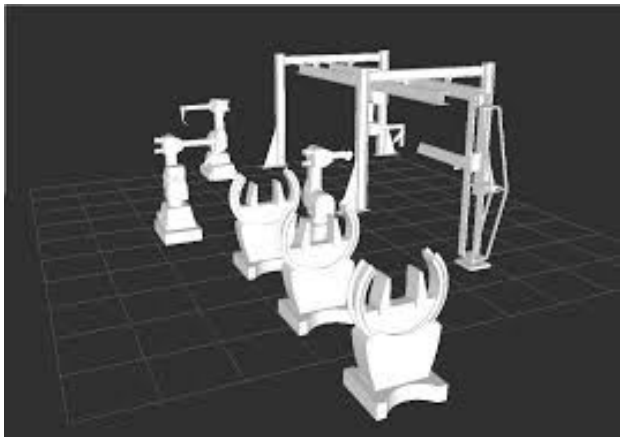


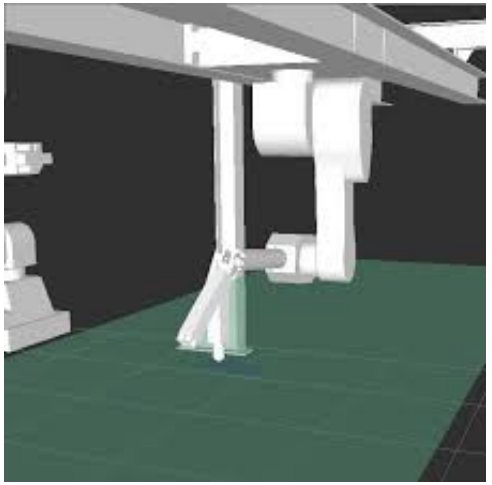
Robots using ROS control

hiDOF

Contracting, technology transfer – USA

- Industrial robots
  - Next-gen steel beam fabrication
  - Plasma cutting, pick and place





Robots using ROS control

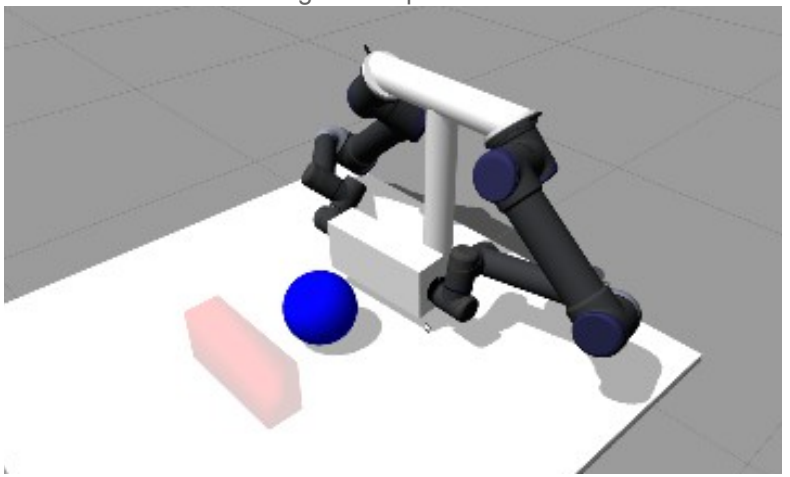
LAAS – Gepetto team

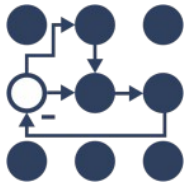
Motion of anthropomorphic systems – France

- PR2
  Implement stack of tasks

source: Willow Garage Flickr photostream

- 2x UR-5 (simulation)
  Manipulation testbed





Robots using ROS control

PAL Robotics

Service robotics – Spain

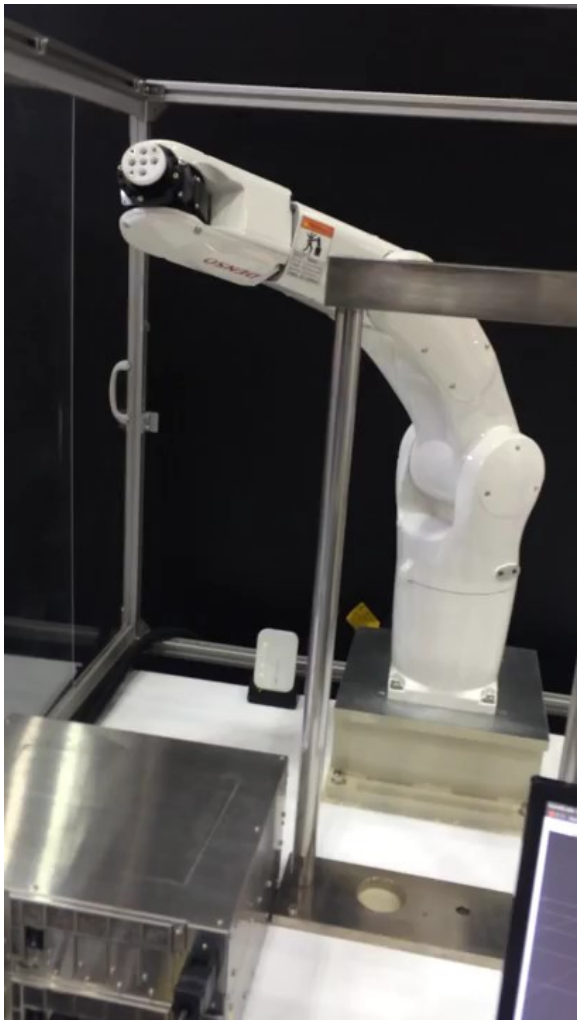
- PMB2
  50kg payload mobile base
- Stockbot
  Robotic inventory solution
- REEM
  Humanoid service robot
- REEM-C
  Biped research robot

Robots using ROS control

TORK

Open source robotics – Japan

- Denso VS060 (WIP)
  Industrial manipulator



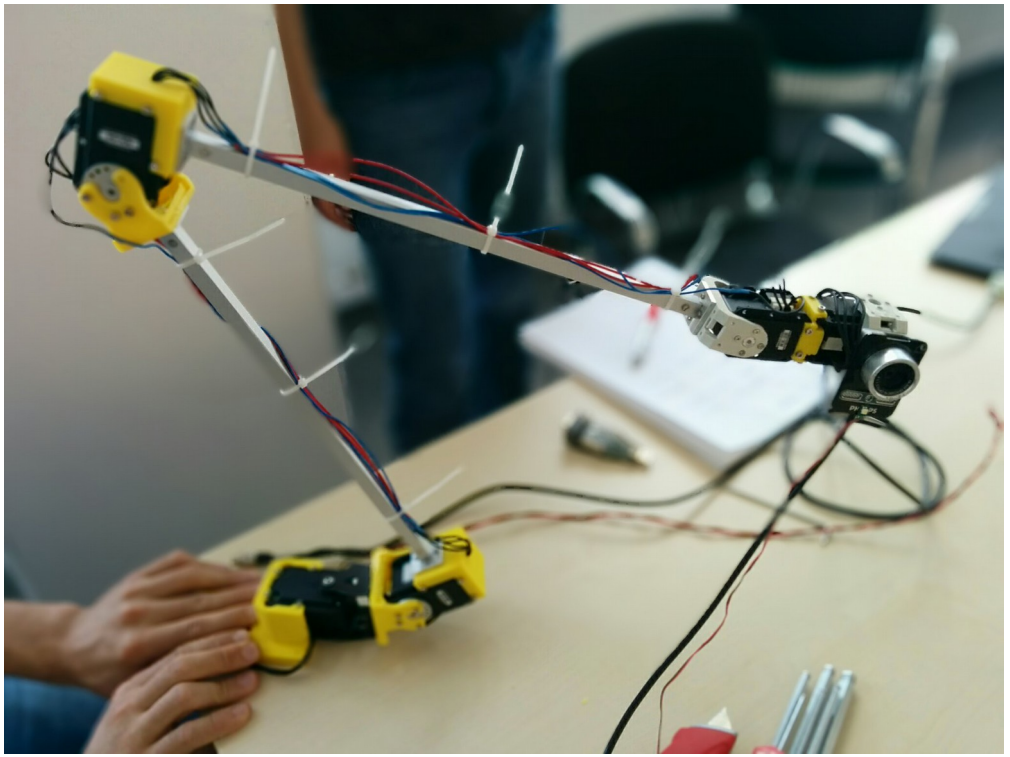
ros_control + ROS-I

Robots using ROS control

TU Darmstadt

Search and rescue – Germany

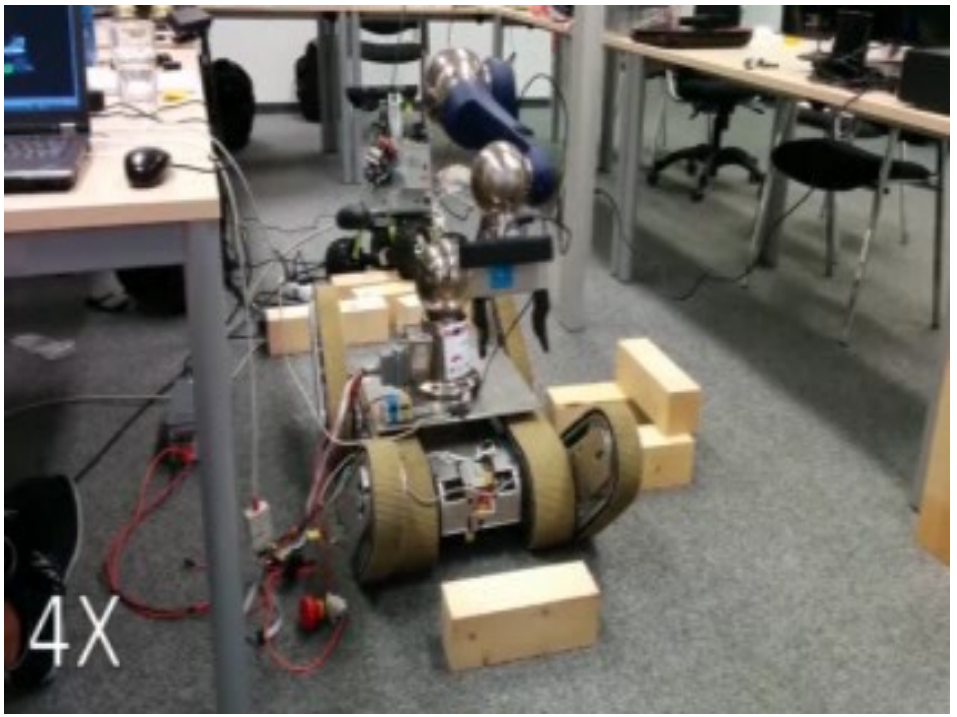
- Dynamixel-based 3D printed arm
  Camera arm control

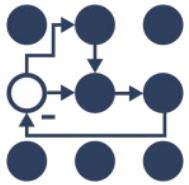


Robots using ROS control

Joint workshop of team TEDUSAR,
TU Darmstadt,
TU Graz,
Search and rescue / manipulation – International

- Schunk LWA4P
  6dof manipulator





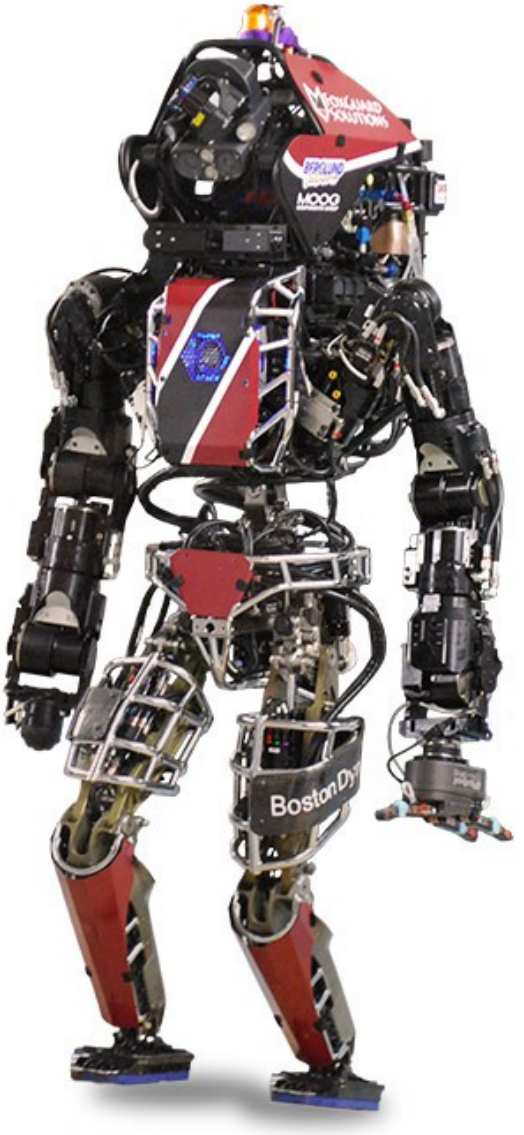
Robots using ROS control

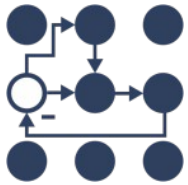
Team VIGIR

- TORC Robotics
- TU Darmstadt
- Virginia Tech
- Oregon State University

Search and rescue – USA, Germany

- ATLAS (WIP)
  Hydraulic biped robot

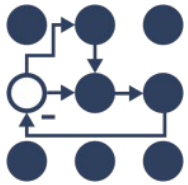




What's next?

ros_control 1.0

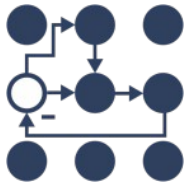
- features
  - library of **robots** and **controllers**
  - composable robots
  - controllers with multiple hardware interfaces
  - joint mode switching
  - multiple controller update rates
- general
  - stabilize APIs
  - user and dev documentation
  - high test coverage, everywhere



What's next?

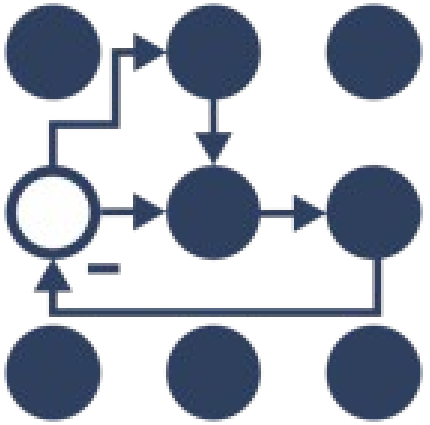
ros_control 2.0

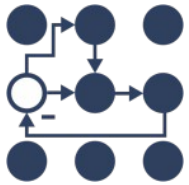
- more flexibility
  - functional / data blocks (ie. controller chaining)
  - lift periodic + serialized controllers requirement
  - dynamic interfaces
- dependency compromise
  - build upon something that exists (no effort duplication)
  - keep dependencies light
  - learning curve: not steeper



Get involved

- **code** github.com/ros-controls
- **SIG** ros-sig-robot-control
- **answers.ros.org** please tag your questions





Acknowledgments

Jonathan Bohren **Dave Coleman** **Wim Meussen**

- Lukas Bulwahn
- Sachin Chitta
- Paul Dinh
- Enrique Fernández
- Kelsey Hawkins
- Austin Hendrix
- Igor Kalevatykh
- Bence Magyar
- Luca Marchionni
- Paul Mathieu
- Victor Mayoral
- Felix Messmer
- Vijay Pradeep
- Mike Purvis
- Jim Rothrock
- Hilario Tomé

Questions?