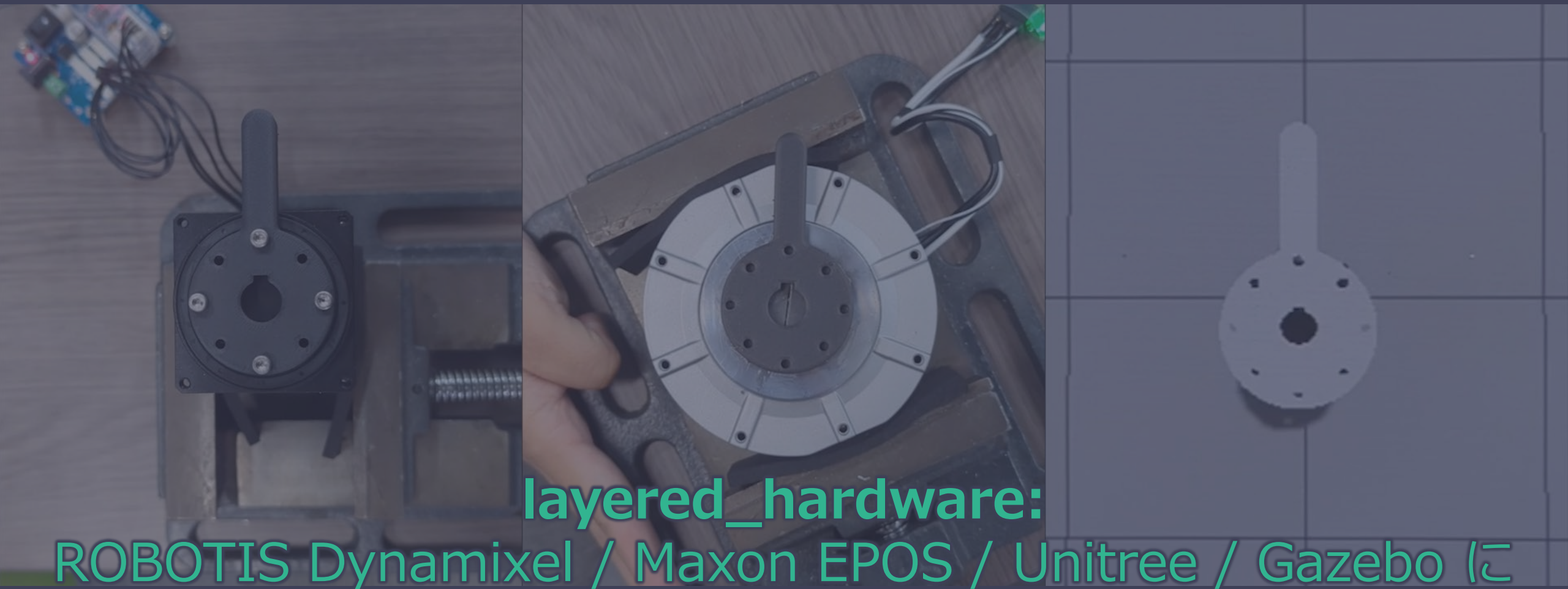


layered_hardware: ROBOTIS Dynamixel / Maxon EPOS / Unitree / Gazebo に対応したプラグイン式 ros2_control

岡田 佳都　小熊 一矢（東北大学）

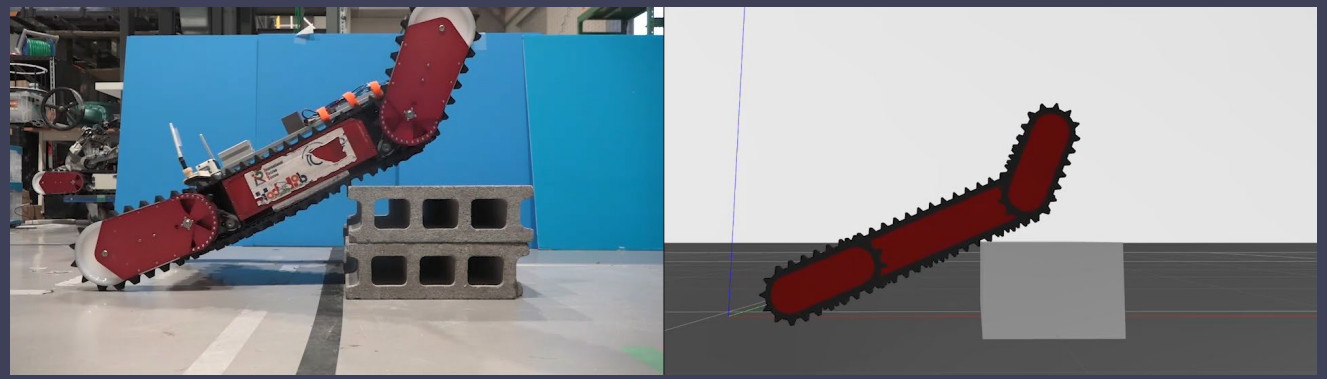
自己紹介｜岡田佳都

- 東北大学 タフ・サイバーフィジカルAI研究センター 准教授
- yoshito-okada

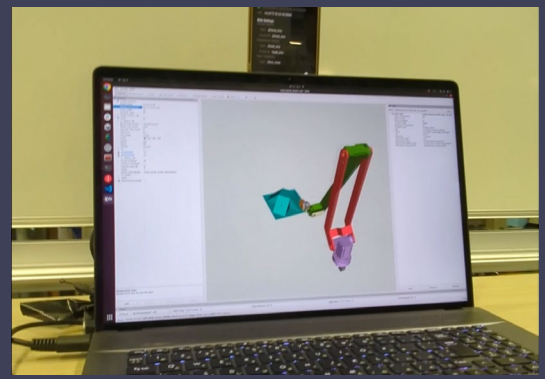
災害対応ロボットの操縦支援
[Okada+, 2010]

球殻ドローンによるインフラ点検
[水谷+, 2013]

姿勢・任意情報を実世界に埋め込む複合マーカ
[Okada+, 2021]

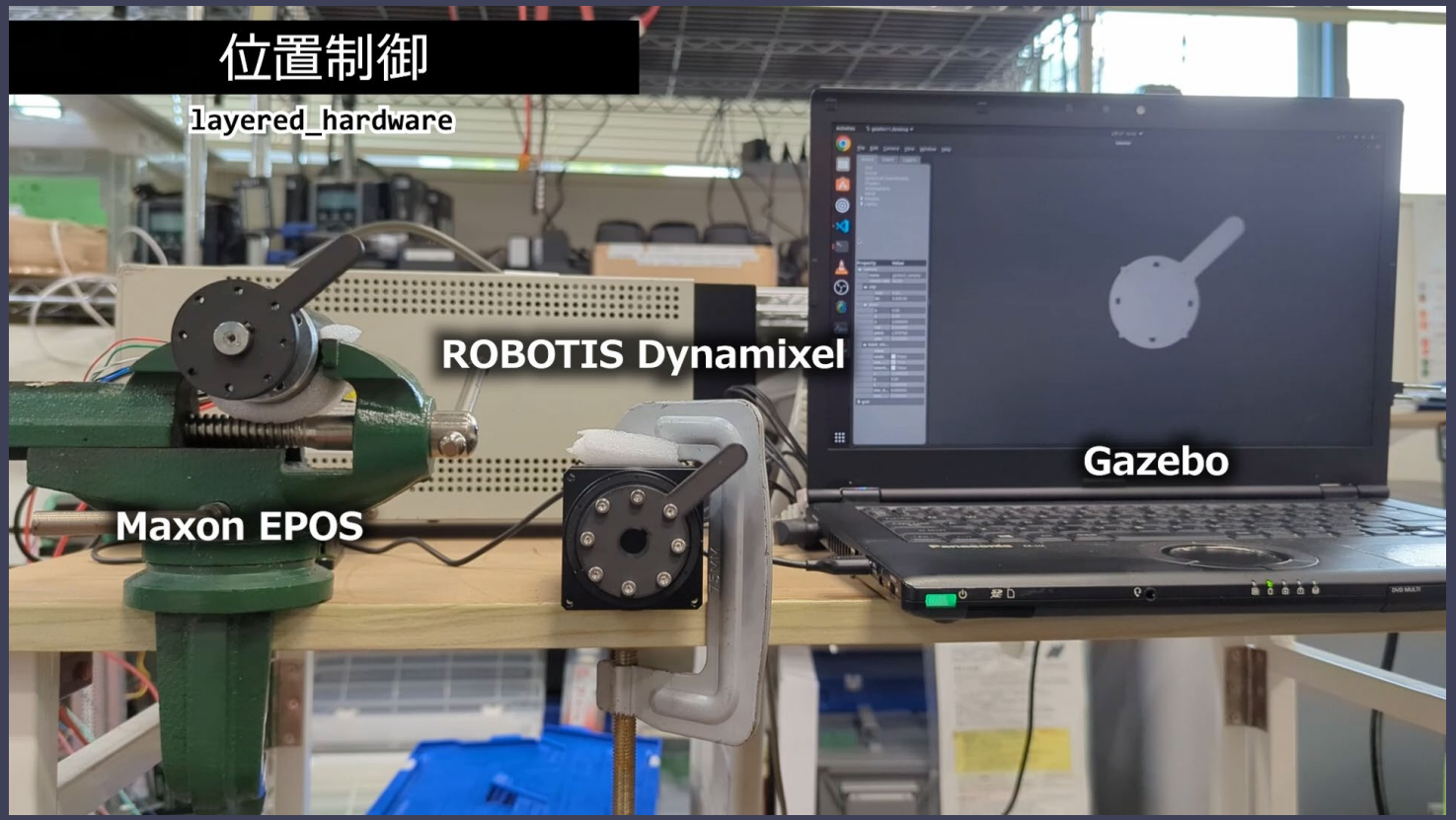


グローサ付きクローラの軽量シミュレーション
[Okada+, 2020]



AirPods 顔位置計測に基づく裸眼立体視 RViz
[Okada, 2023]

本日のメイン:
プラグイン式 ros2_control **layered_hardware**



ROS 1版の複数アクチュエータ・複数制御モード対応のデモ

※ 本日はROS 2版＆新アクチュエータ対応の発表！

yoshito-n-students/
layered_hardware

※ ROS 2版も同リポで近日公開予定

TL;DR

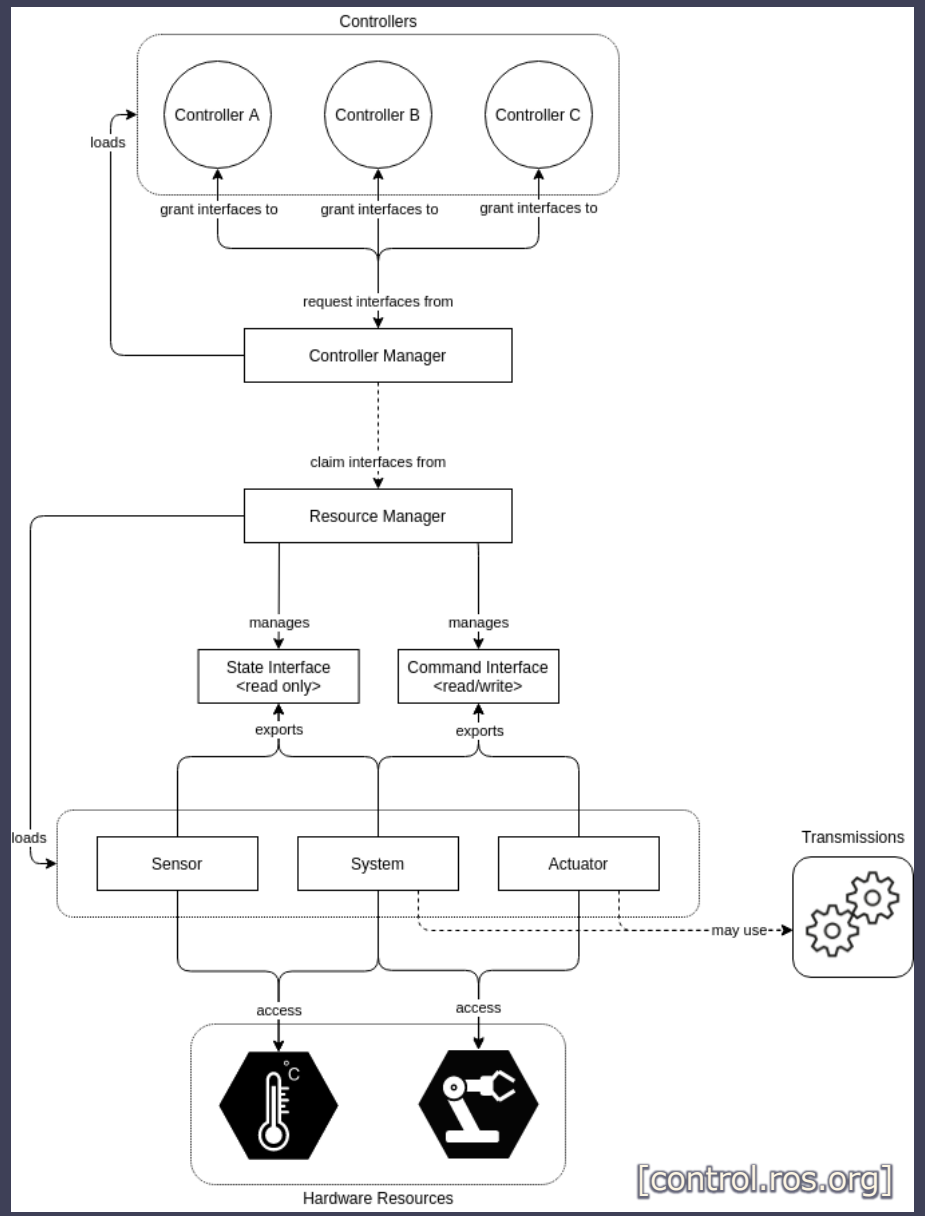
ros_control 勢目線では

ros2_control のモジュールの独立性を維持しながら実装できる新しいフレームワークを作りました，という話

一般 ROS ユーザ勢目線では

Dynamixel も Maxon Epos も Unitree も Ignition Gazebo も ros2_control で回せる便利パッケージができました，という話

そもそも ros2_control って何？

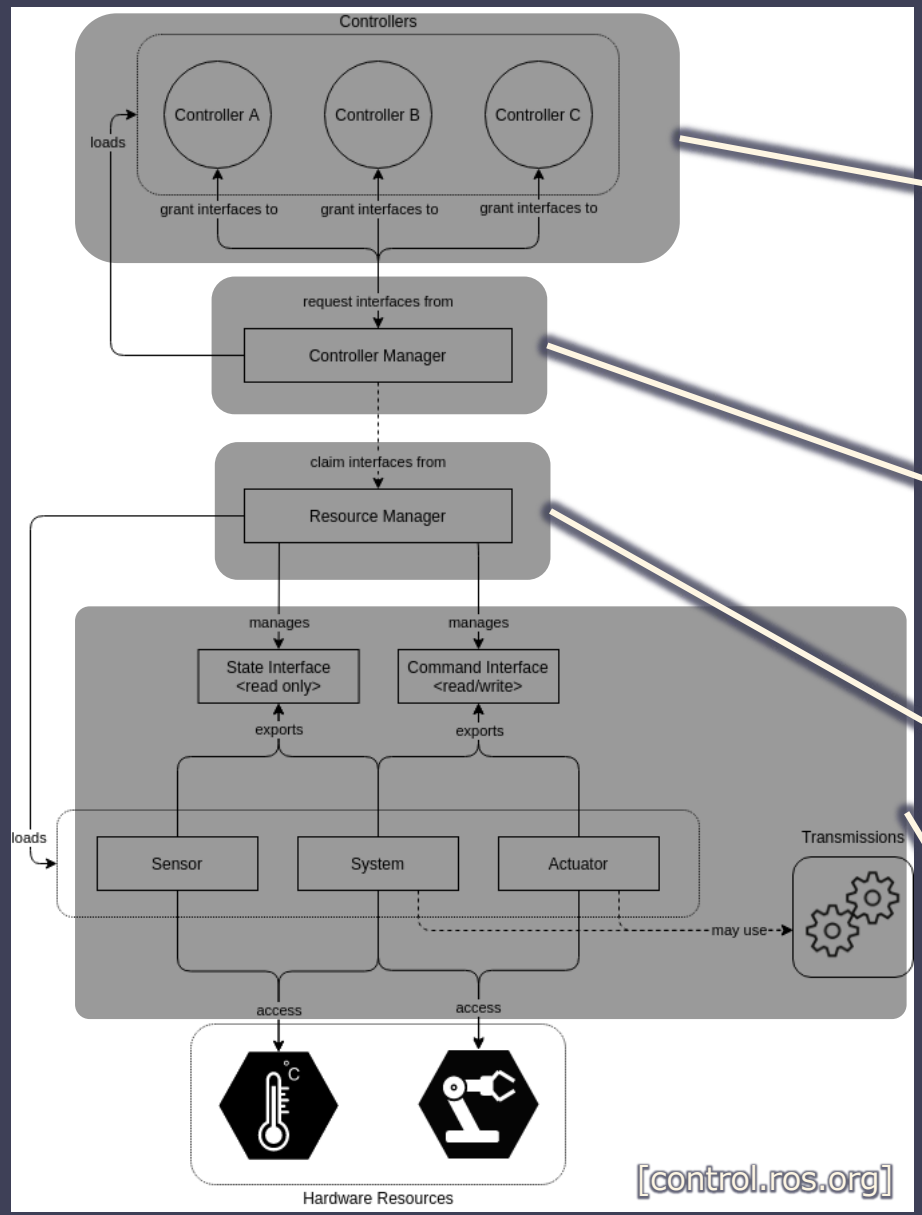


- 制御ロジックを実装した **Controller** と
  アクチュエータやセンサと通信する **System** を
  1つのプロセス内で実行するためのフレームワーク
- **Controller** と **System** の間は
  プロセス内の値渡しであるため
  Topic 通信のオーバーヘッドがない
- **Controller** と **System** はプラグイン形式なので
  自分のロボットに合わせて自作できる
  （ただしC++に限る）
- 使いこなせれば便利
  ＆公式ドキュメントも頑張ってはいるが
  要素が多くとっつきにくい

なのでまずわかりやすく説明します！

ちょっと長いけど本編の伏線でもあるのでご勘弁

誰でもワカル!? ros2_control とはなにか



登場人物

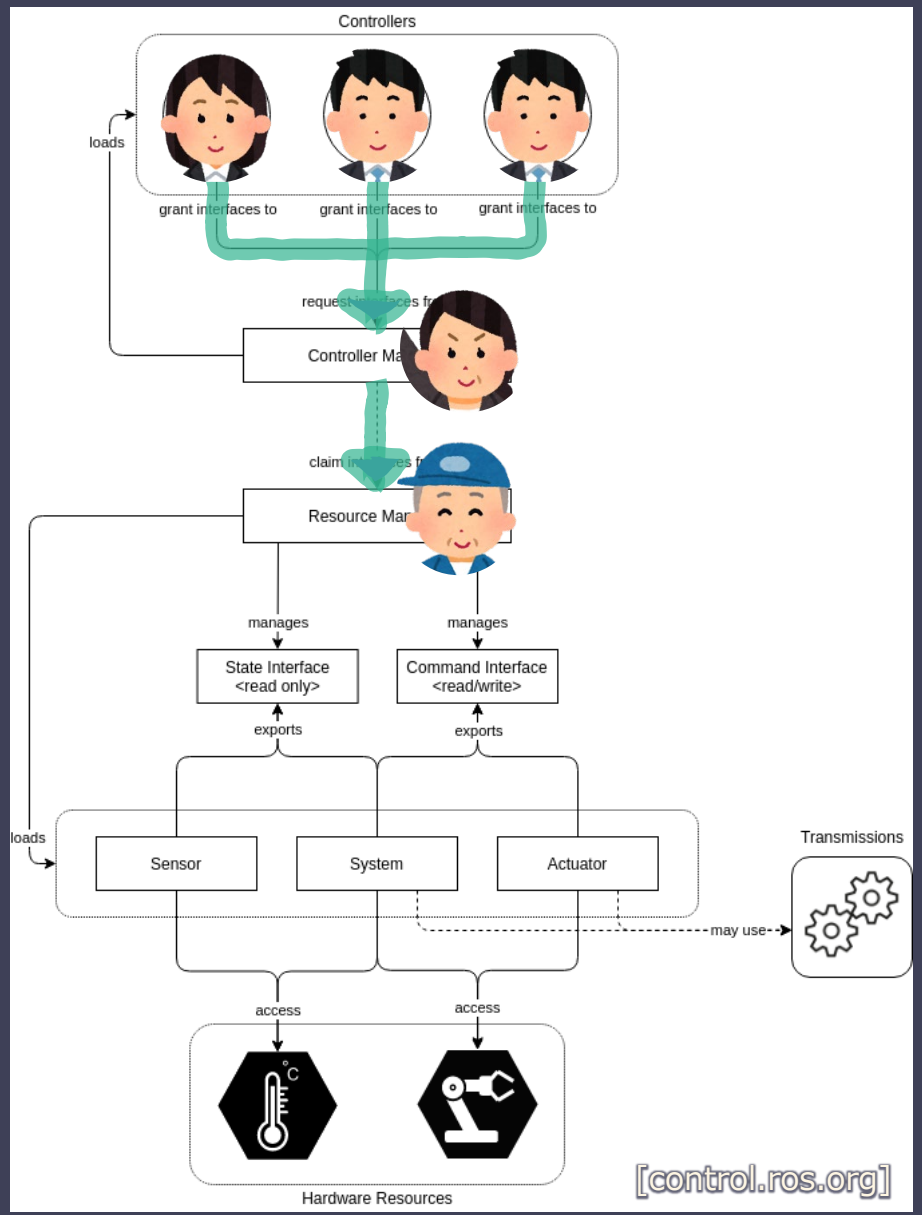
Controller たち
- 他 Node と Topic 通信
- 制御ロジックを実装
- System の Command, State を読み書き

ControllerManager
- Controller をプラグインとしてロード
- Controller たちを管理

ResourceManager
- System をプラグインとしてロード
- System を管理

System
- アクチュエータやセンサと通信
- Controller に Command, State を提供

誰でもワカル!? ros2_control とはなにか



ros2 run controller_manager ros2_control_node するとこうなる 開始編1/3

Contoller A

shoulder_joint/velocity を読みたいなぁ

```
InterfaceConfiguration state_interface_configuration()
```

Contoller B

finger_joint/position に書き込みたいなぁ

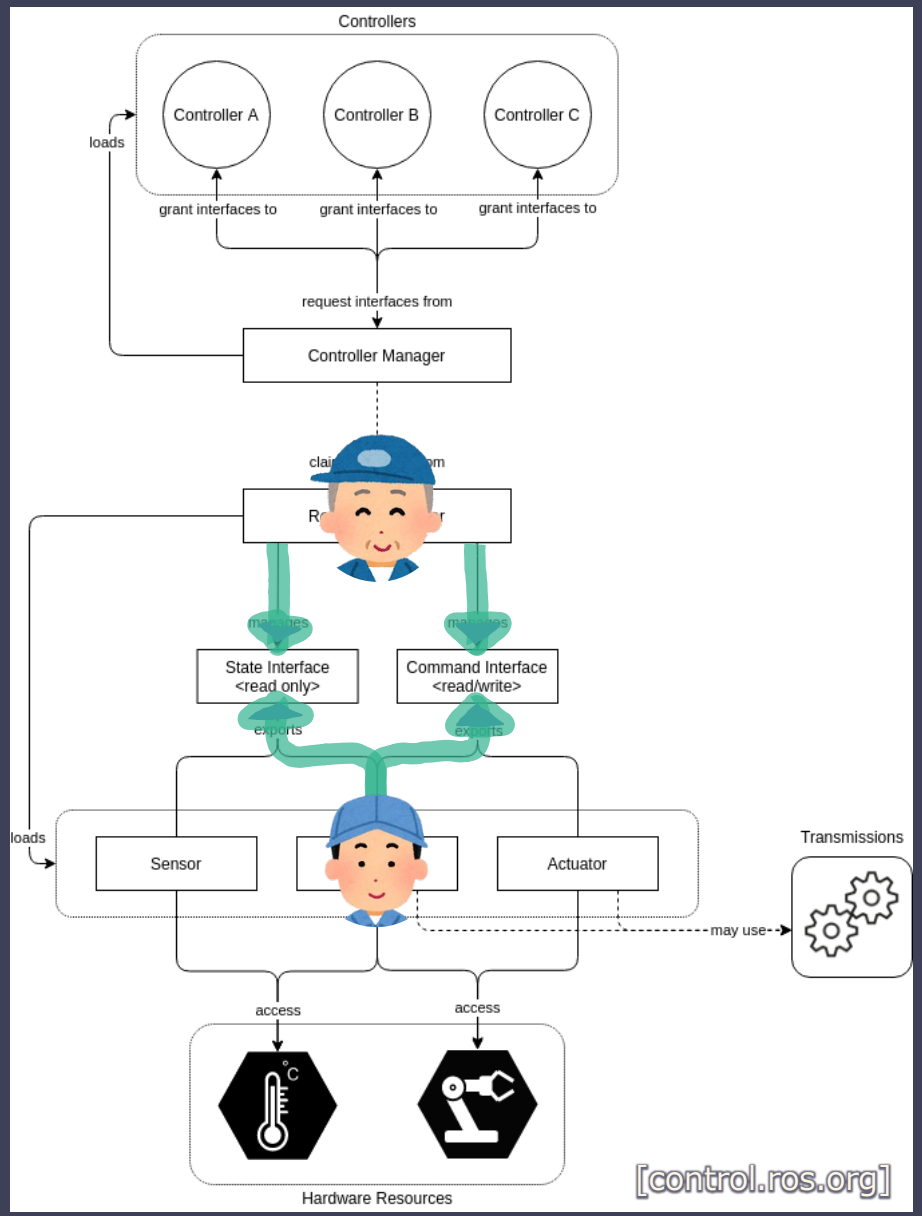
```
InterfaceConfiguration command_interface_configuration()
```

ContollerManager

わかった
探してくる

阿吽の呼吸

誰でもワカル!? ros2_control とはなにか



ros2 run controller_manager ros2_control_node するとこうなる 開始編2/3

ResouceManager
何持ってる？

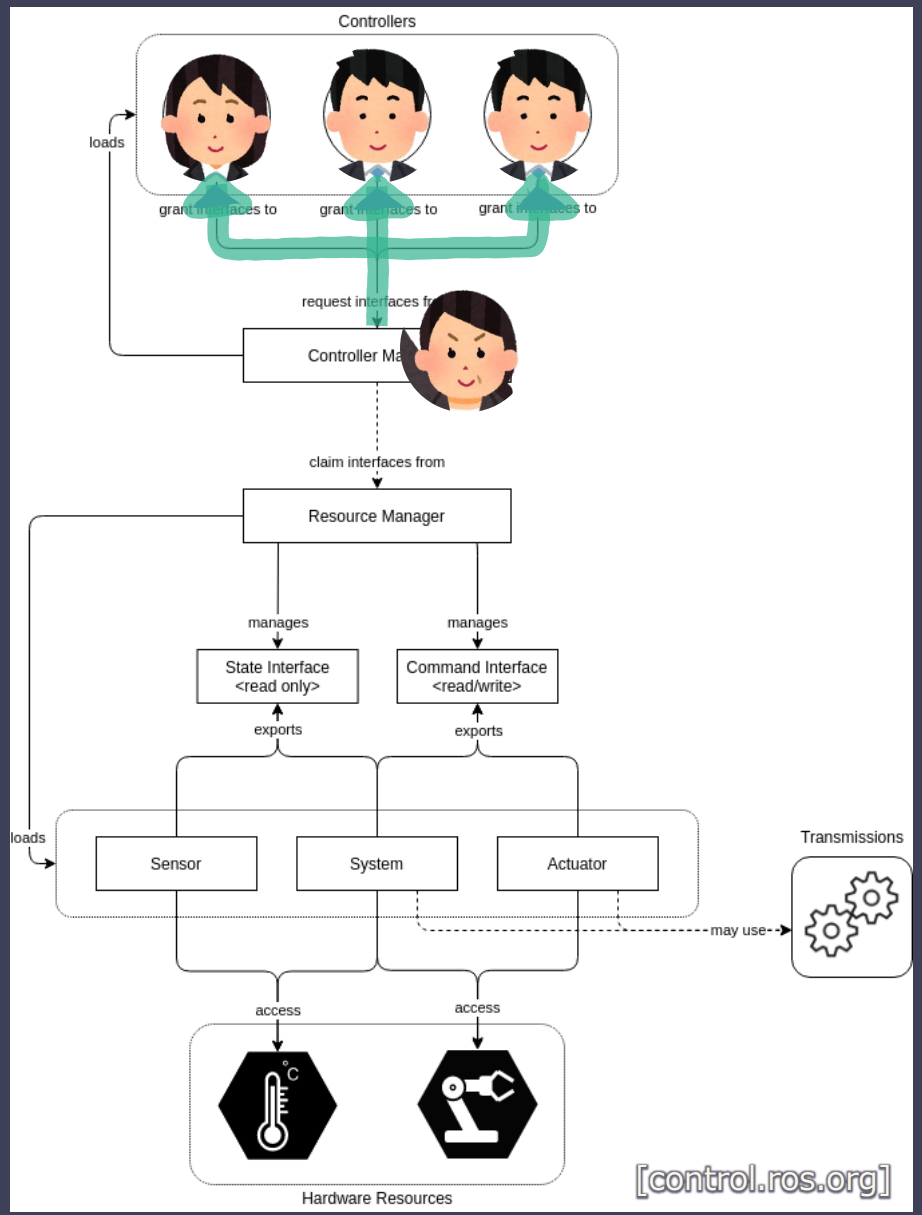
System
shoulder_joint/position, velocity, … ありますよ
あと finger_joint/position, effort もあります

```
std::vector<StateInterface> export_state_interfaces()
```

```
std::vector<CommandInterface> export_command_interfaces()
```

ResouceManager
わかった
それ預かるわ

誰でもワカル!? ros2_control とはなにか



ros2 run controller_manager ros2_control_node するとこうなる 開始編3/3


```
void assign_interfaces(
  std::vector<hardware_interface::LoanedCommandInterface> && command_interfaces,
  std::vector<hardware_interface::LoanedStateInterface> && state_interfaces);
```

System から **Controller** に
State や Command の参照が渡り
制御ループが周期実行できる！

というのが左図の意味

誰でもワカル!? ros2_control とはなにか

ros2 run controller_manager ros2_control_node するとこうなる 周期実行編

ResouceManager
読んで！

System
センサやアクチュエータから状態を読み出します～

```
return_type read(const rclcpp::Time & time, const rclcpp::Duration & period)
```

ControllerManager
更新かけて！

Controller たち
コマンドを更新します～

```
return_type update(const rclcpp::Time & time, const rclcpp::Duration & period)
```

ResouceManager
書いて！

System
センサやアクチュエータにコマンド伝えます～

```
return_type write(const rclcpp::Time & time, const rclcpp::Duration & period)
```

誰でもワカル!? ros2_control とはなにか

ros2 service call /controller_manager/switch_controller

するとこうなる モード切替編1/2

service_caller
あのControllerクビにして
代わりにあいつ採用できる？

ControllerManager
検討します

ResourceManager
こうなりそうだけど対応できる？

System
まあ何とか

```
virtual return_type prepare_command_mode_switch(
  const std::vector<std::string> & /*start_interfaces*/,
  const std::vector<std::string> & /*stop_interfaces*/)
```

誰でもワカル!? ros2_control とはなにか

ros2 service call /controller_manager/switch_controller

するとこうなる モード切替編2/2

ControllerManager
異動です！

ControllerManager
完了です

Controller B
今までありがとう。findger_joint/position 返します

Controller C
新人です！finger_joint/effort 欲しいです

service_caller
ご苦労

ControllerManager
異動あったって

System
新人に合わせて finger を effort モードに変えますね～

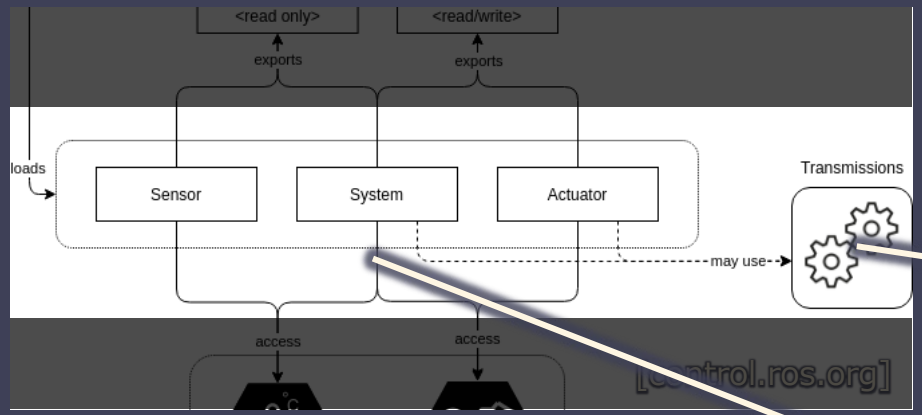
```
virtual return_type perform_command_mode_switch(
  const std::vector<std::string> & /*start_interfaces*/,
  const std::vector<std::string> & /*stop_interfaces*/)
```

ros2_control に残る問題点（私見）

自作Systemを実装する際，**様々なモジュールを組み合わせる**必要がある

→ 素朴に書くと**可読性・メンテナンス性・拡張性**がよろしくない

→ 別のアクチュエータに変えると，**Joint limits や Transmissions の実装も変わってしまう**



頻出モジュールの例

ros2_control /joint_limits

- Joint Command を位置・速度・加速度の制限を満たすように上書きする
- 制限を適用する関数群は開発途中で未リリース（2024年9月時点）

ros2_control /transmission _interface

- ギア比等を使って Joint Command を Actuator Command に変換する
- 逆に，Actuator State を Joint State に変換する

some_vendor /actuator_sdk

- Actuator Command を Actuator に送る
- Actuator State を Actuator から取得する

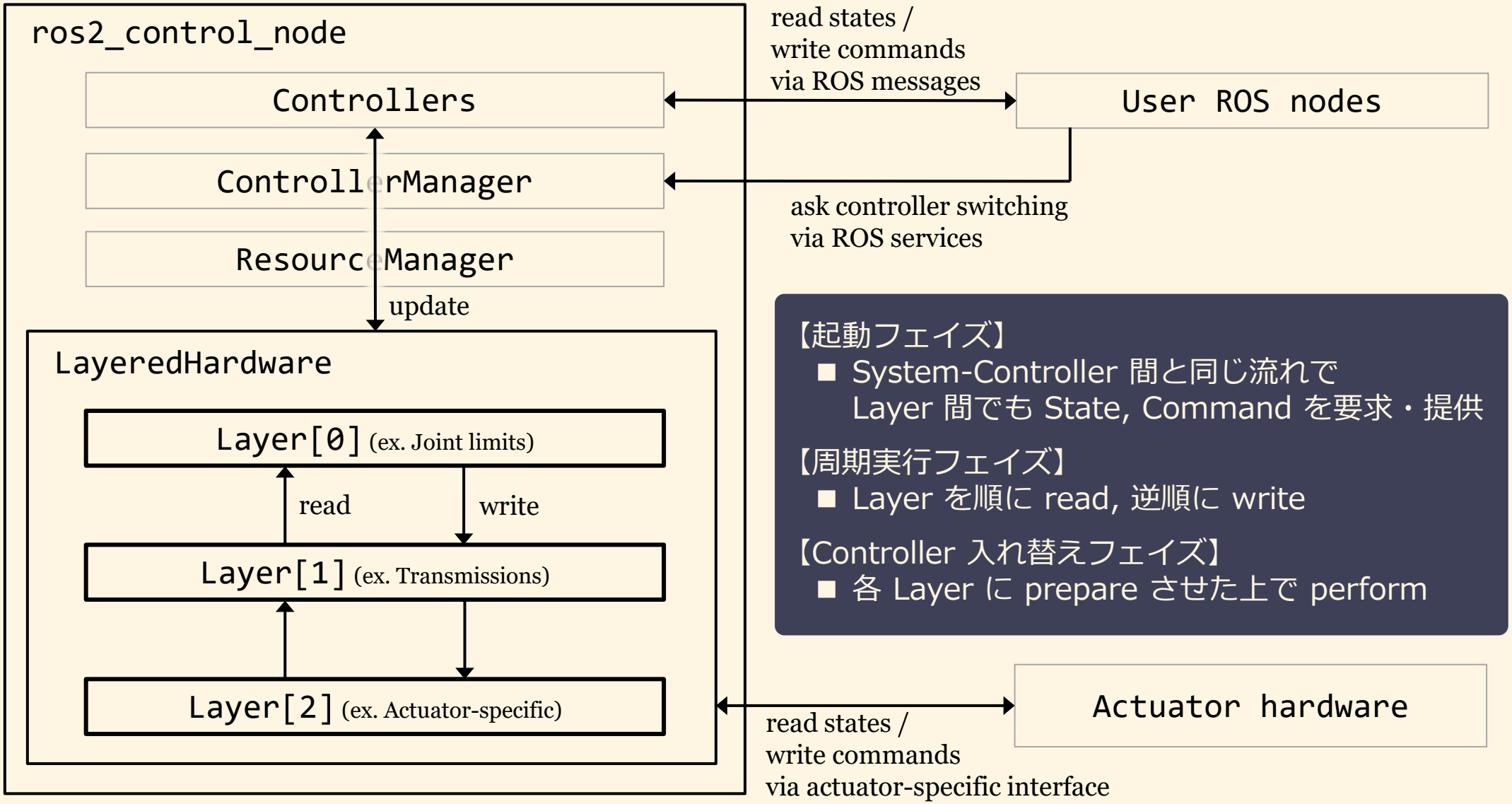
layered_hardware 開発の狙い

モジュールの**独立性を保ちつつ** ros2_control の System を自作したい！
アクチュエータを変えても**他のモジュールの実装は共通にしたい**！

→ **各モジュールをプラグイン化し，**
汎用の System からロードすればいいのでは？

layered_hardware

ROS 2版 layered_hardware のコンセプト

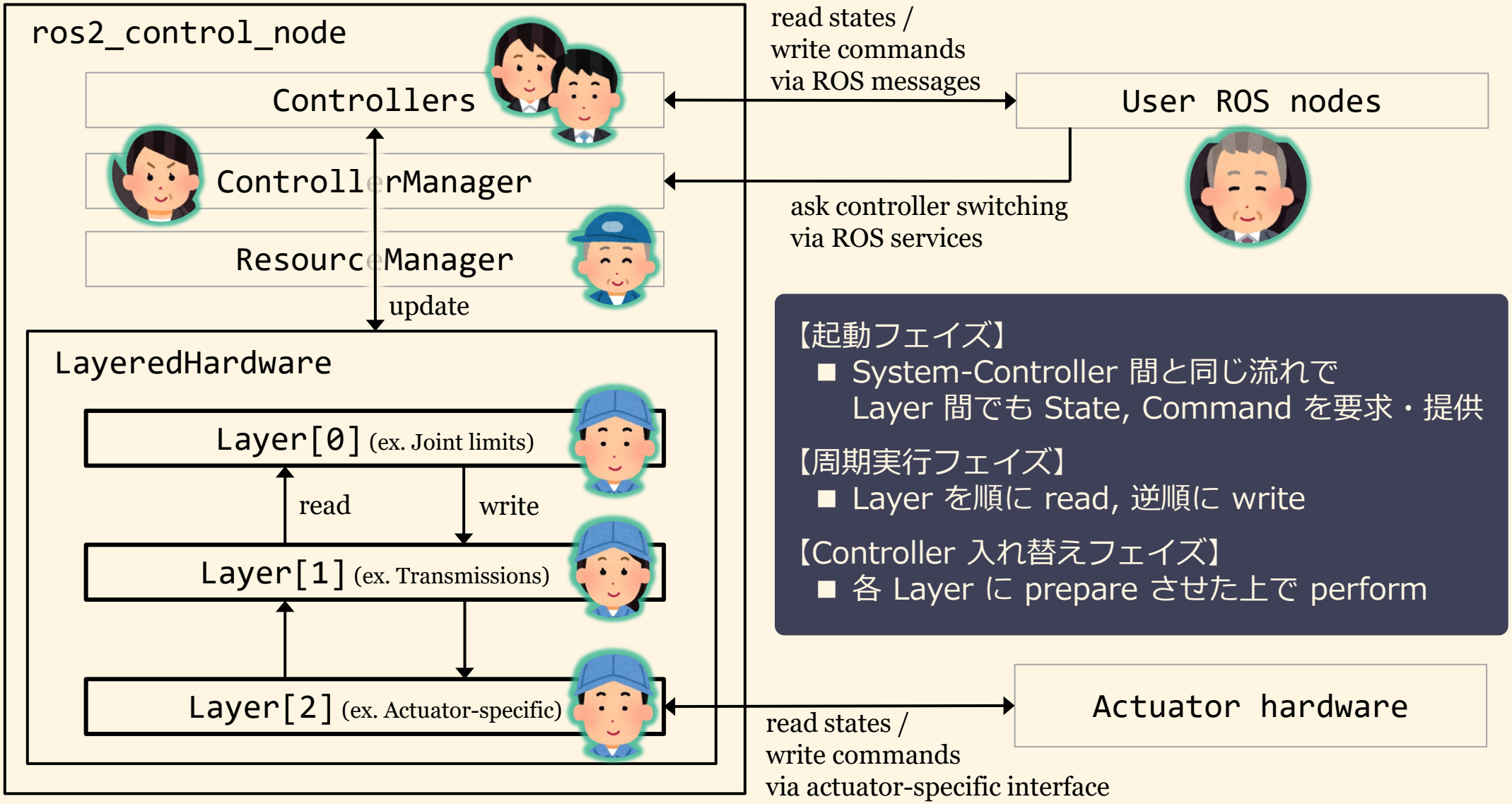


【起動フェイズ】
- System-Controller 間と同じ流れで Layer 間でも State, Command を要求・提供

【周期実行フェイズ】
- Layer を順に read, 逆順に write

【Controller 入れ替えフェイズ】
- 各 Layer に prepare させた上で perform

ROS 2版 layered_hardware のコンセプト（ワカルver.）

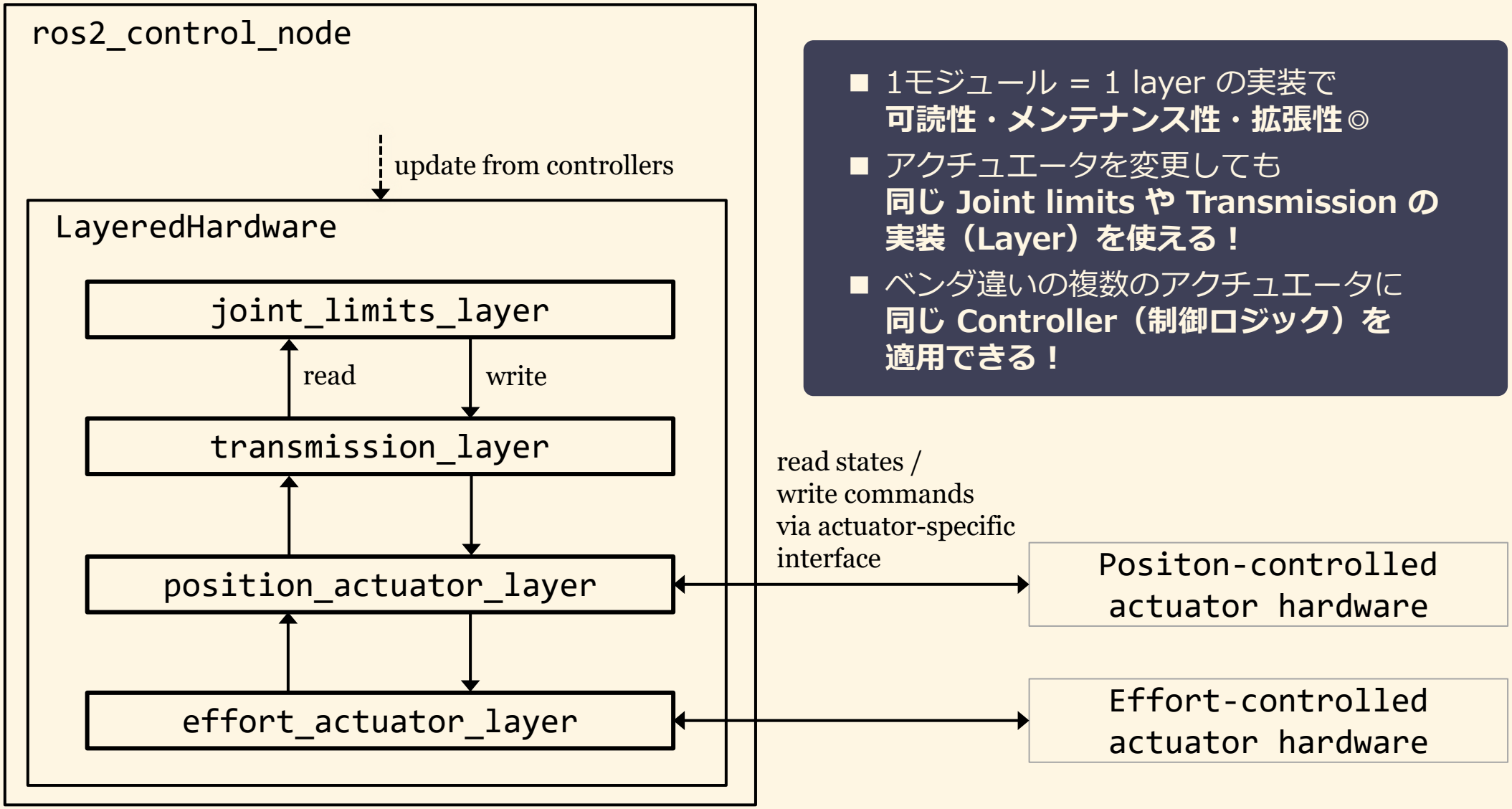


【起動フェイズ】
- System-Controller 間と同じ流れで Layer 間でも State, Command を要求・提供

【周期実行フェイズ】
- Layer を順に read, 逆順に write

【Controller 入れ替えフェイズ】
- 各 Layer に prepare させた上で perform

ROS 2版 layered_hardware の構成例



- 1モジュール = 1 layer の実装で
  可読性・メンテナンス性・拡張性◎
- アクチュエータを変更しても
  同じ Joint limits や Transmission の実装（Layer）を使える！
- ベンダ違いの複数のアクチュエータに
  同じ Controller（制御ロジック）を適用できる！

関連 Pkg ①:

layered_hardware_unitree

- 犬型ロボットで有名な Unitree 社製の
  アクチュエータを回すための Layer を提供
- PC側の接続方式は USB-シリアル
  （e.g. /dev/ttyUSBN）
- **位置制御，速度制御，トルク制御**をサポート
  （オンライン切り替えも可能）
- 速度制限付き位置制御なども
  JointLimitsLayer の利用でサポートできる予定
  （ros2_control/joint_limits のリリース待ち）

昨日見たやつ
（本発表と無関係
だけど凄い）

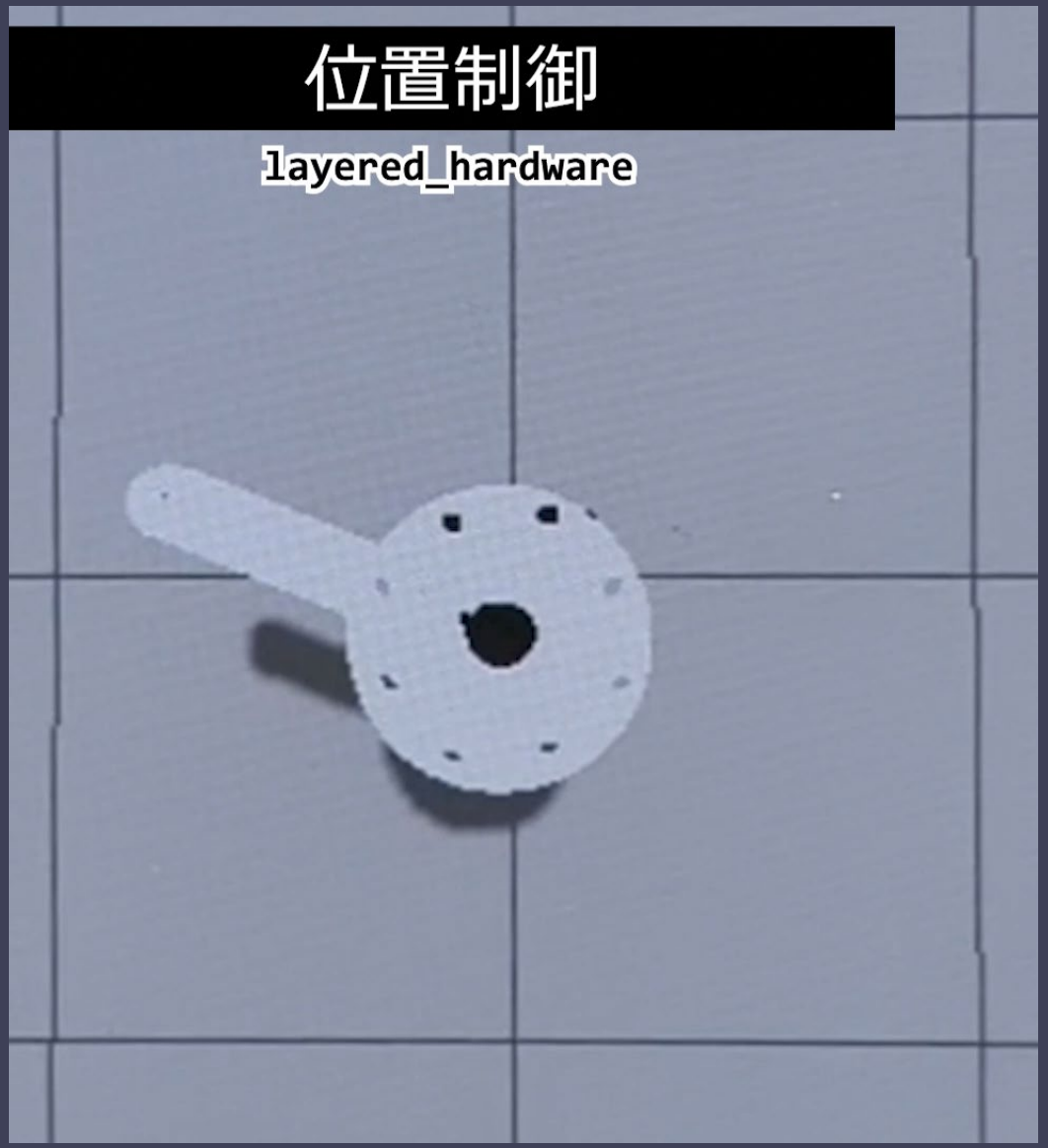
関連 Pkg ②:

layered_hardware_dynamixel

- ROBOTIS 社製の Dynamixel シリーズの
  アクチュエータを回すための Layer を提供
- バックエンドである
  公式 dynamixel-workbench がサポートする
  モデルは全て回せるはず
- PC側の接続方式は USB-シリアル
  （e.g. /dev/ttyUSBN）
- **{位置，速度，トルク}制限付き**
  {位置，速度，トルク} 制御をサポート
  （モデルが対応している制御モードに限る。
  オンライン切り替えも可能）

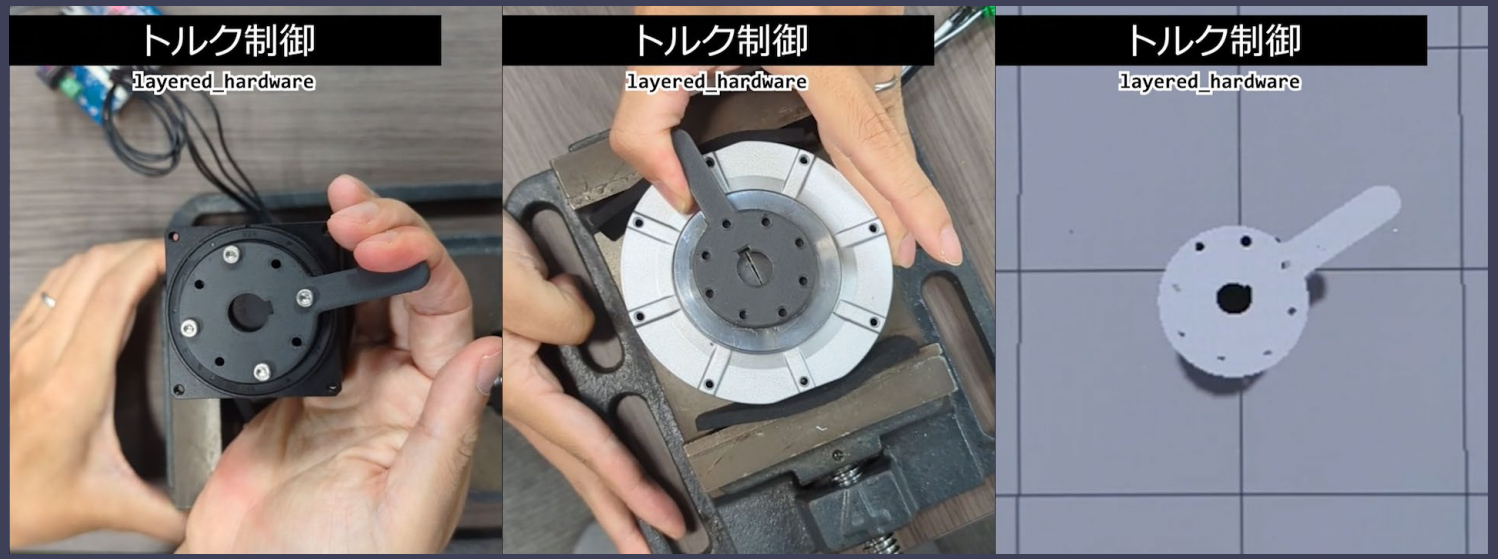
関連 Pkg ③:

layered_hardware_ign



- Ignition Gazebo (not Gazebo Classic)の Joint を回すための Layer を提供
- **位置制御，速度制御，トルク制御**をサポート（オンライン切り替えも可能）
- 速度制限付き位置制御なども JointLimitsLayer の利用でサポートできる予定（ros2_control/joint_limits のリリース待ち）

ROS 2 Humble 版リリースを Star してお待ちください 🙏



yoshito-n-students/
layered_hardware

※ ROS 2版も同リポで近日公開予定

高度な例: 16DoF (クローラ8 / アーム6 / グリッパ2)

TOHOKU UNIVERSITY

TCPAI
Tough Cyberphysical AI Research Center
Tohoku University

layered_hardware

Maxon EPOS
ROBOTIS Dynamixel
Unit

Gazebo

yoshito-n-students/
layered_hardware

ありがとうございました！ロボットでテスト後リリースします 🙏