ETロボコン向けTOPPERS活用セミナー
2024/06/22

# ETロボコン向け開発環境の紹介とSPIKE-RTの環境構築手順

樋山一樹
（南山大学 / TOPPERS）

# 目次

- 自己紹介
- RasPike・SPIKE-RT・RasPike-ROSの位置付けを整理
- SPIKE-RTの環境構築方法
- ROS 2とRasPike-ROSの紹介

# 自己紹介

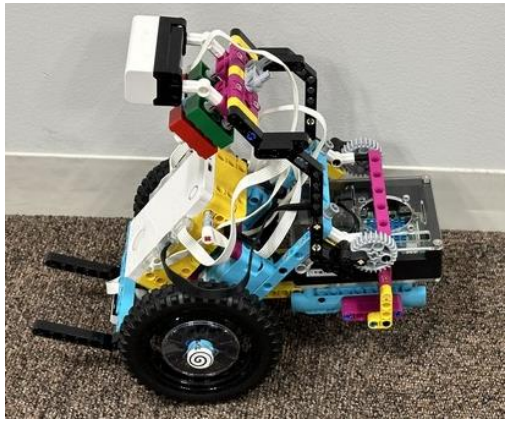
**樋山一樹（ひやま　いつき）**

- 愛知県在住
- 南山大学 M1
- 学部3年時に研究室に配属されて以来，組込みシステムを中心に学習中
  - 研究室HP：https://honda-lab1.sakura.ne.jp
- 研究室配属後にTOPPESの活動に参加

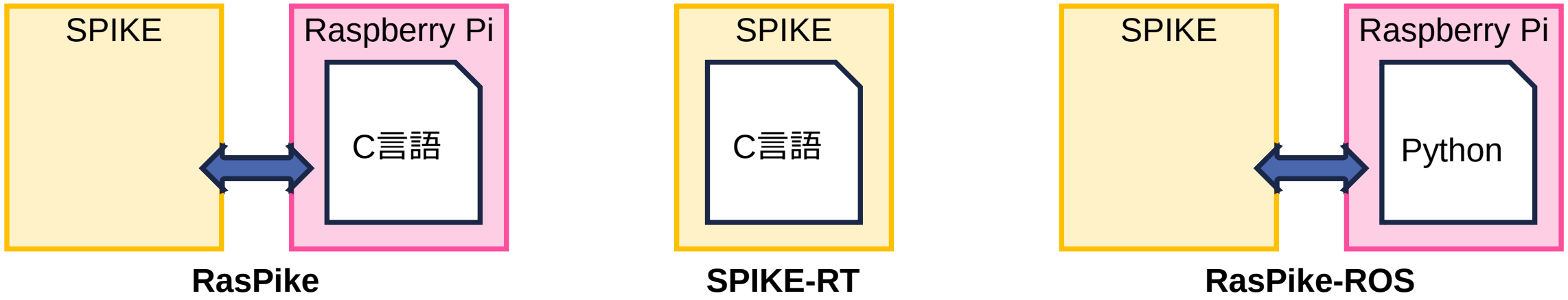
# 目次

- 自己紹介
- RasPike・SPIKE-RT・RasPike-ROSの位置付けを整理
- SPIKE-RTの環境構築方法
- ROS 2とRasPike-ROSの紹介

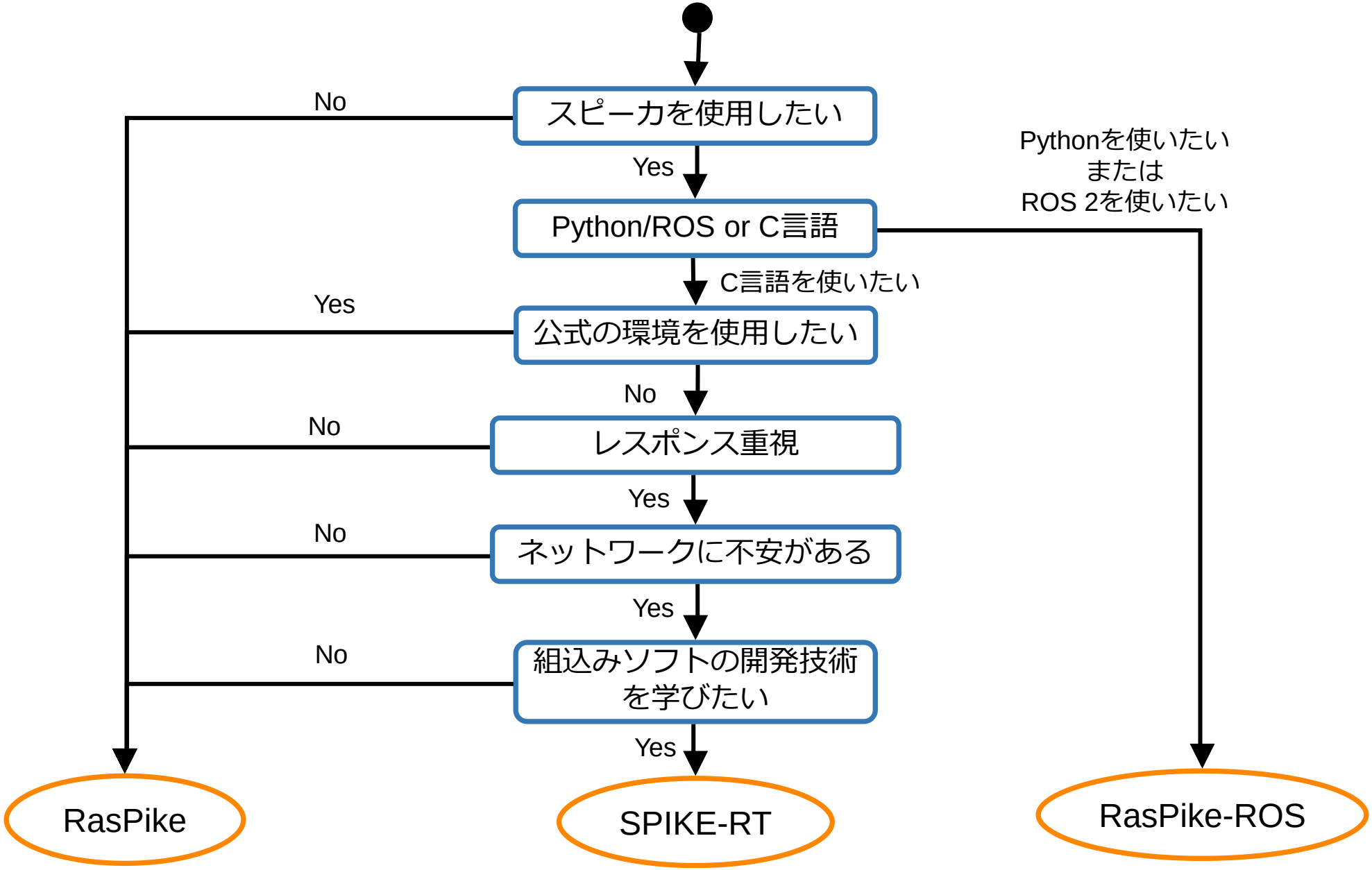
# 各プラットホームの特徴



| ーーー | 開発言語 | ROS 2 | 開発環境 | ネットワーク | レスポンス |
|---|---|---|---|---|---|
| **RasPike** | C言語 | × | RaspberryPi | 必要 | △ |
| **SPIKE-RT** | C言語 | × | ホストPC | 不要 | ○ |
| **RasPike-ROS** | Python | ○ | RPI / ホストPC | 必要 | △+ |



# どのプラットホームが向いているか

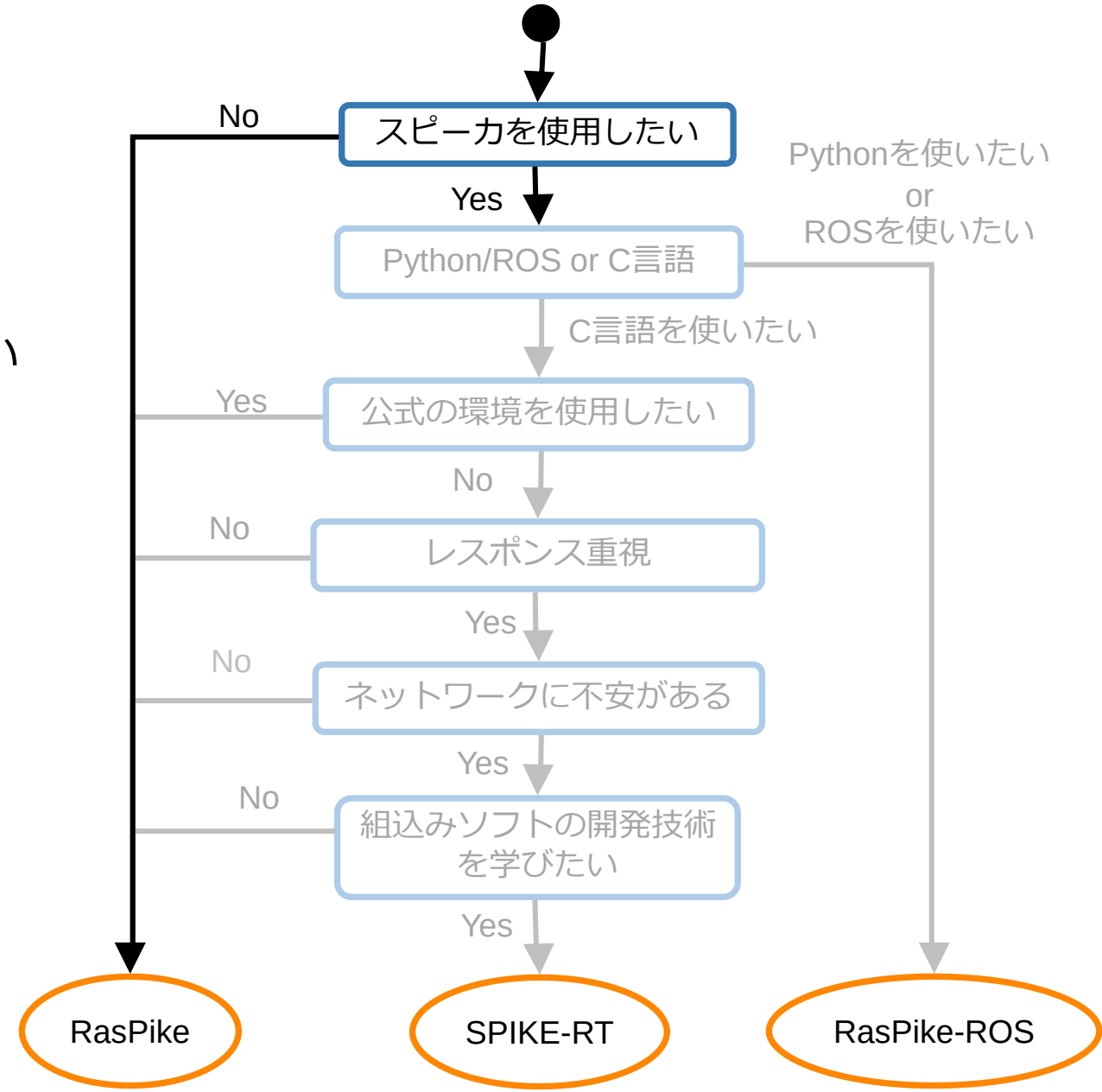


# スピーカを使用したい

- RasPikeはスピーカが非対応
  - デバッグ等でスピーカを使用したい場合はSPIKE-RT・RasPike-ROS

- 利用可能モジュールの制限が最も少ないものはSPIKE-RT

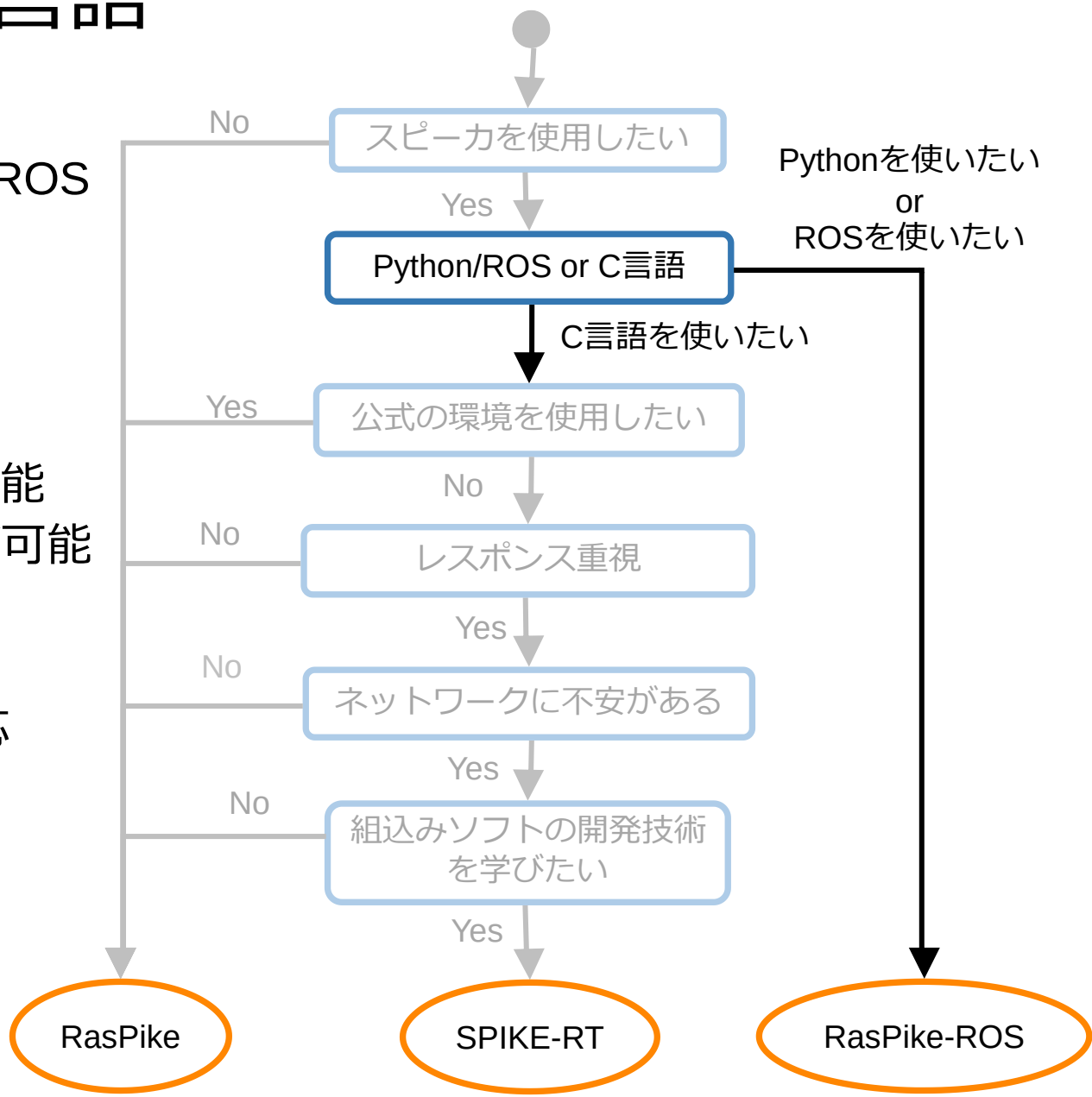
| ーーー | スピーカ |
|---|---|
| **RasPike** | 利用不可 |
| **SPIKE-RT** | 利用可 |
| **RasPike-ROS** | 利用可 |



# Python/ROS 2またはC言語

- RasPike-ROSのみPythonで開発
  - Pythonを使用したい場合はRasPike-ROS
- SPIKE-RTとRasPikeはC言語で開発
- RasPike-ROSはROS 2を使用
  - カメラやPC等をROS 2通信で接続可能
  - ホストPC上でアプリの開発・実行が可能
  - ROS 2の説明は後述
- マルチタスク（スレッド）はどれも対応

| ーーー | 開発言語 | ROS 2 | 開発環境 |
|---|---|---|---|
| **RasPike** | C言語 | × | RaspberryPi |
| **SPIKE-RT** | C言語 | × | ホストPC |
| **RasPike-ROS** | Python | ○ | RPI / ホストPC |

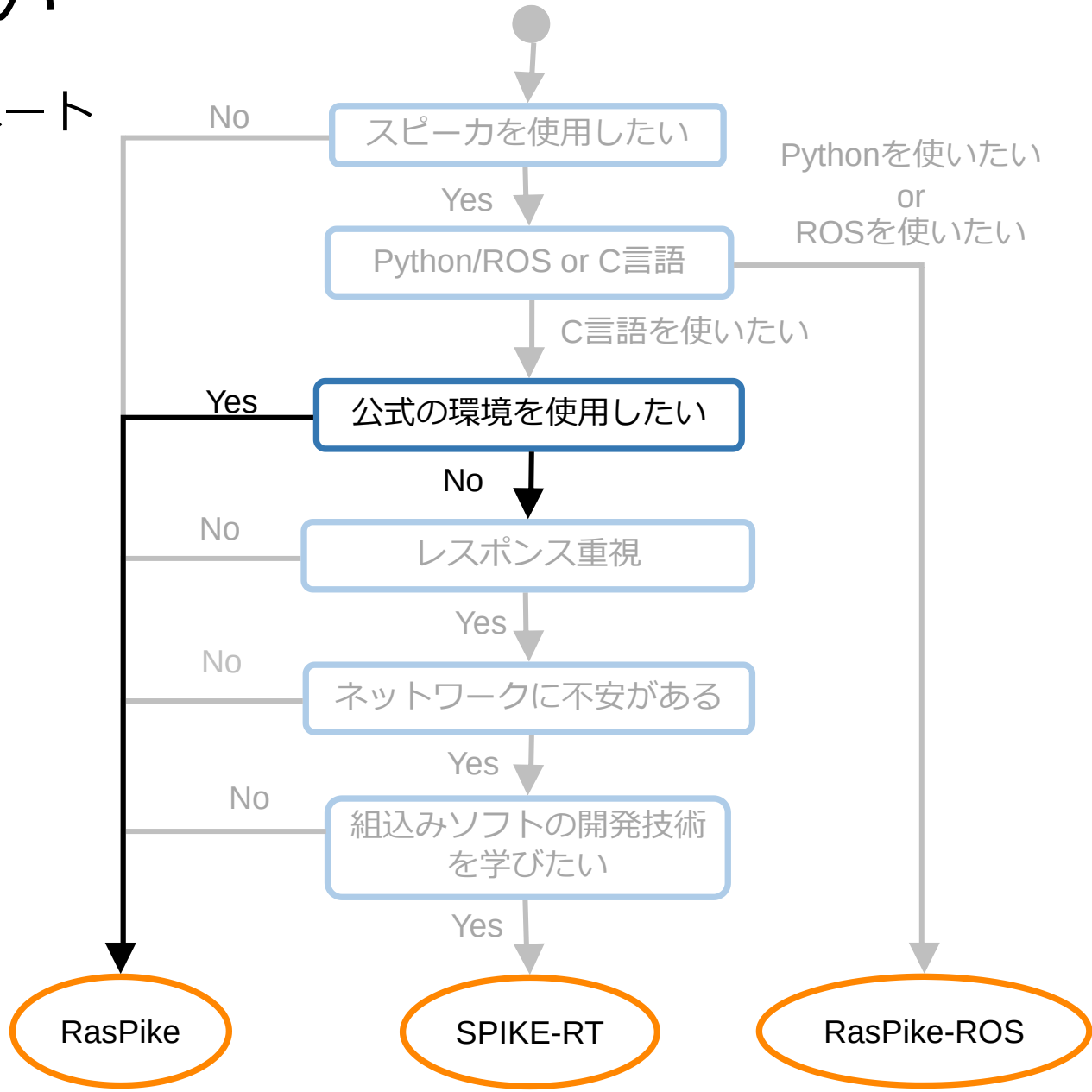


# 公式の環境を使用したい

- SPIKE-RT・RasPike-ROSは公式でサポートされていない

- 利用しても規約違反ではない
  - Raspberry Piは必ず搭載しなければならない

SPIKE-RT・RasPike-ROSの質問はSlackコミュニティで対応



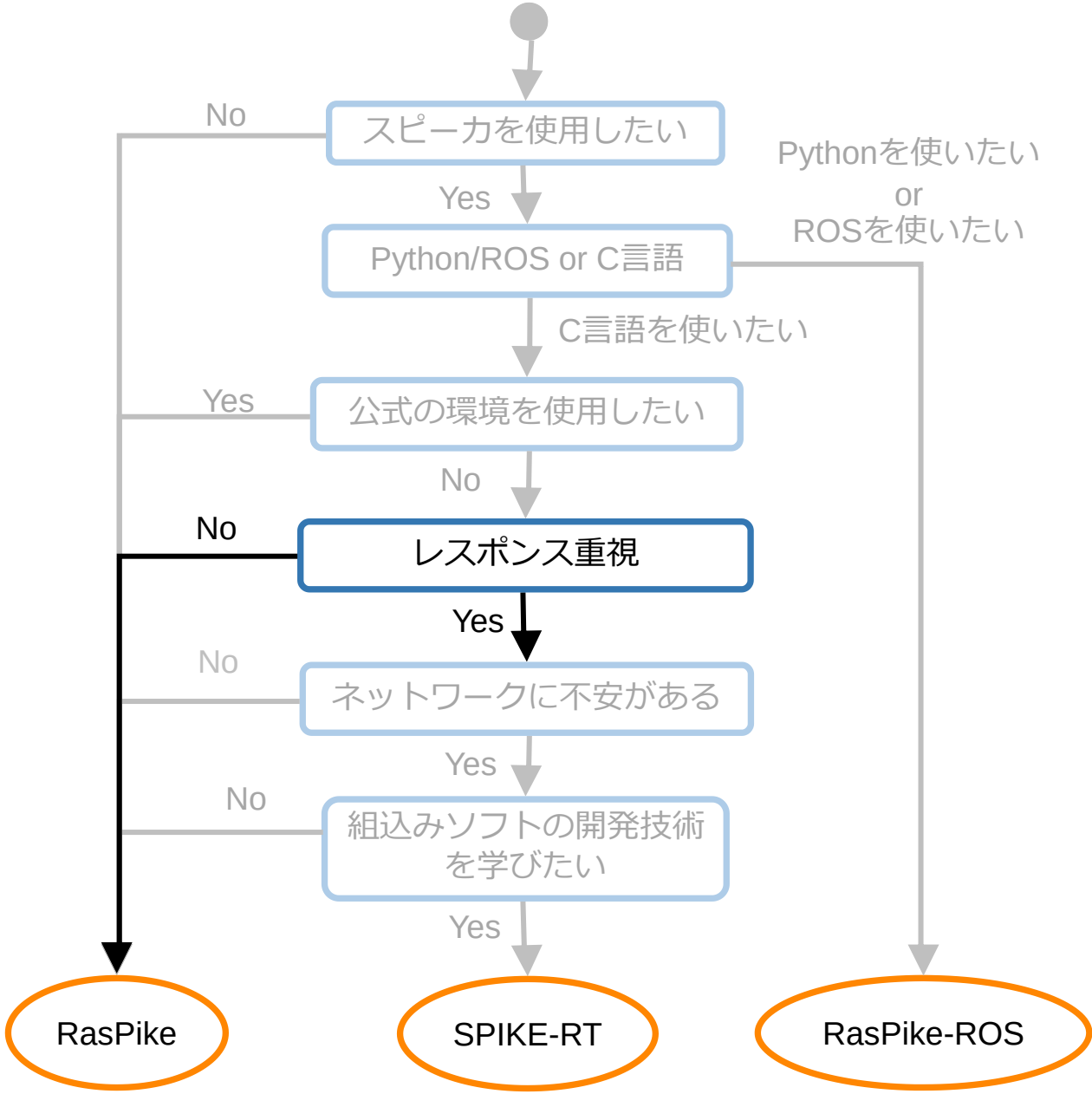
# レスポンス重視

- SPIKE-RTは軽量かつ高速
  - RTOS

- SPIKE-RTはRaspberry Piと通信する必要がない
  - 通信による遅延が発生しない

- RasPike-ROSはROS 2(DDS)を使用

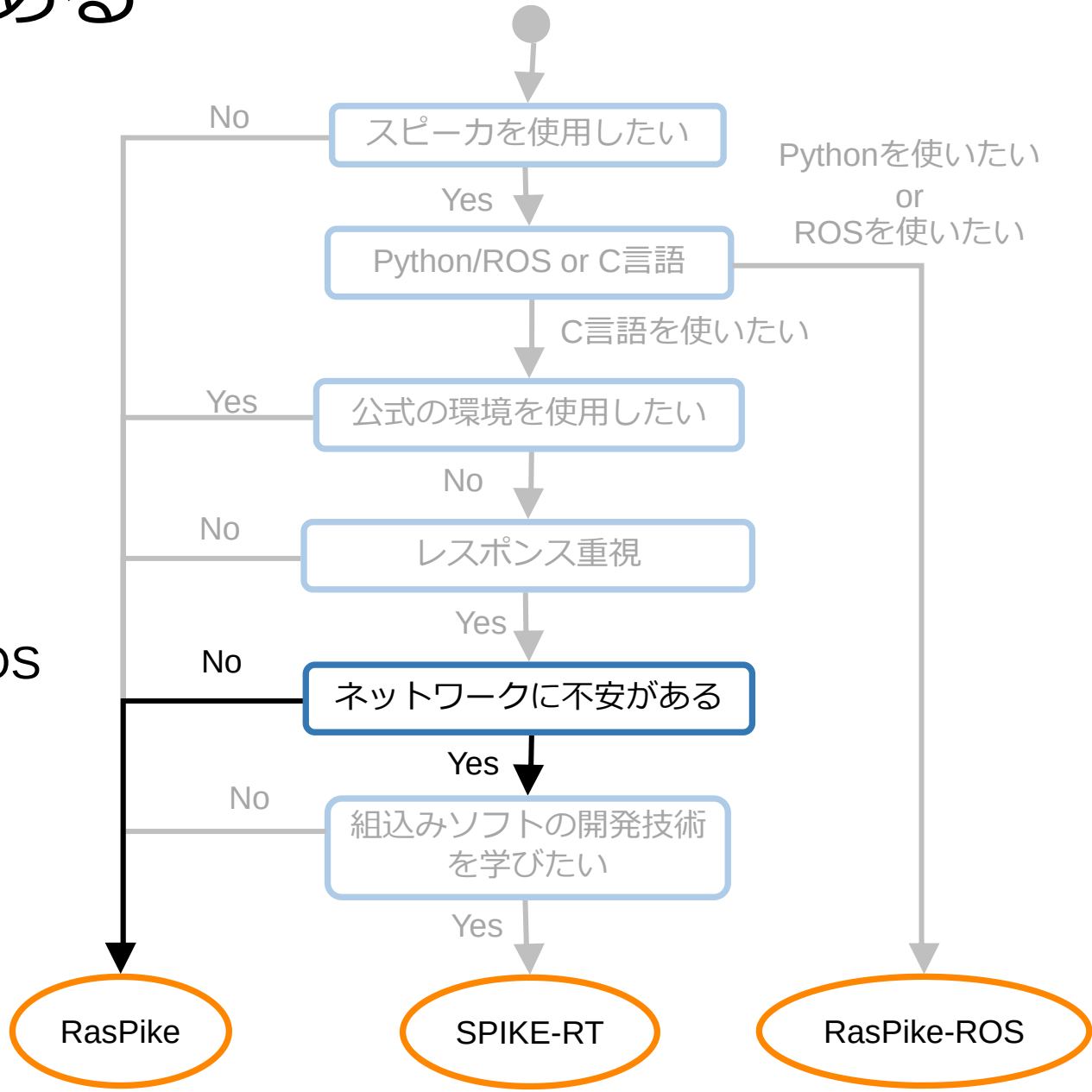
| ーーー | レスポンス |
|---|---|
| **RasPike** | △ |
| **SPIKE-RT** | ○ |
| **RasPike-ROS** | △+ |



# ネットワークに不安がある

- RasPikeはRaspberry Piの操作が必要
  - ネット経由，UART，HDMI，etc…
  - sshの場合はIP固定が必要…
- SPIKE-RTはSPIKE単体で動作
  - 電源を投入したらプログラムが始動
- RasPike-ROSはホストPC上でアプリ開発と実行が可能
  - RPI操作が不安→SPIKE-RT，RasPike-ROS
  - ネットが不安→SPIKE-RT

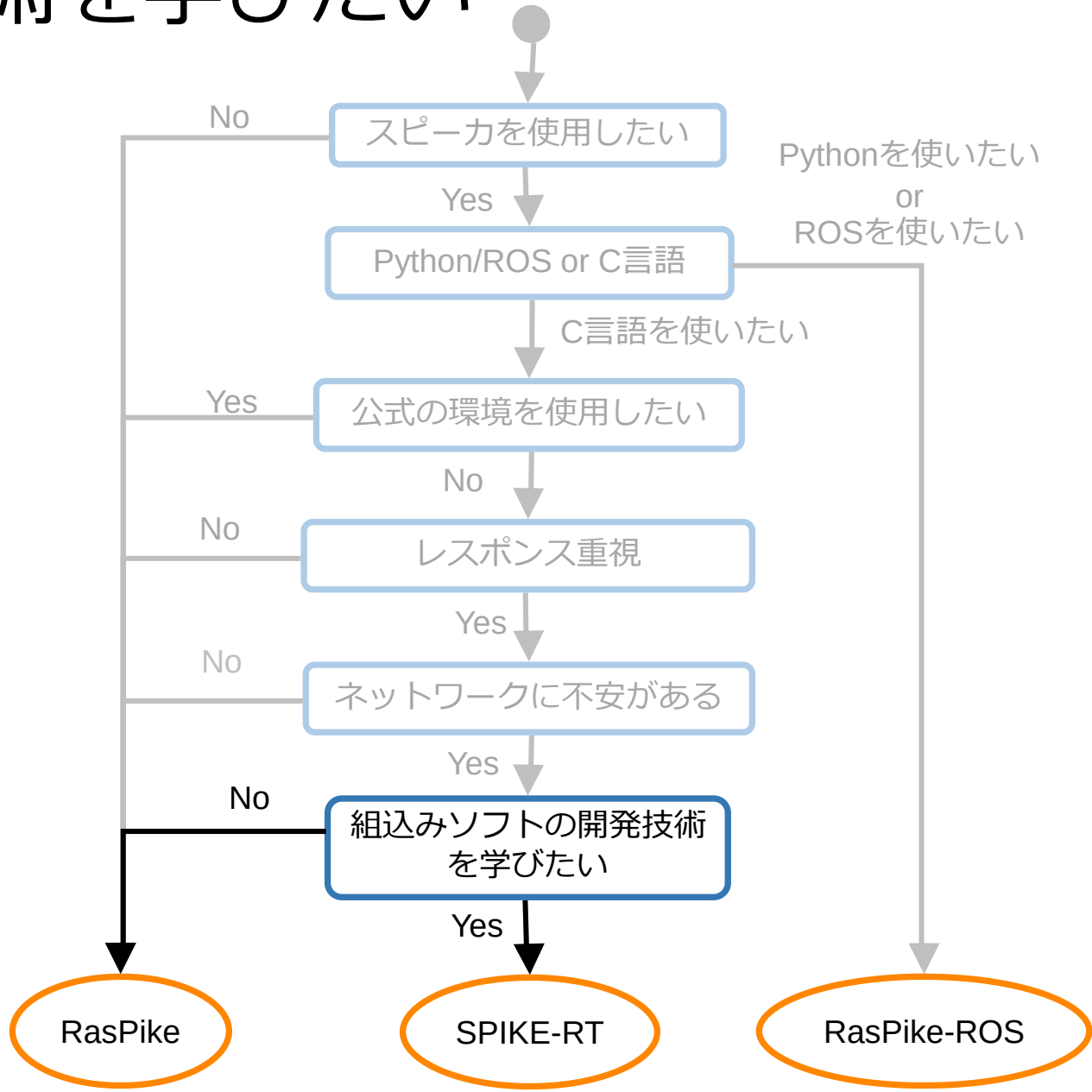
| ーーー | 開発環境 | ネットワーク |
|---|---|---|
| **RasPike** | RaspberryPi | 必要 |
| **SPIKE-RT** | ホストPC | 不要 |
| **RasPike-ROS** | RPI / ホストPC | 必要 |



# 組込みソフトの開発技術を学びたい

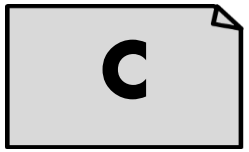
- SPIKE-RTではクロス開発を体験できる
  - ホストPCで開発・ビルド
  - ターゲット(SPIKE)に書き込む

- 純粋なRTOSを使用したプログラミング

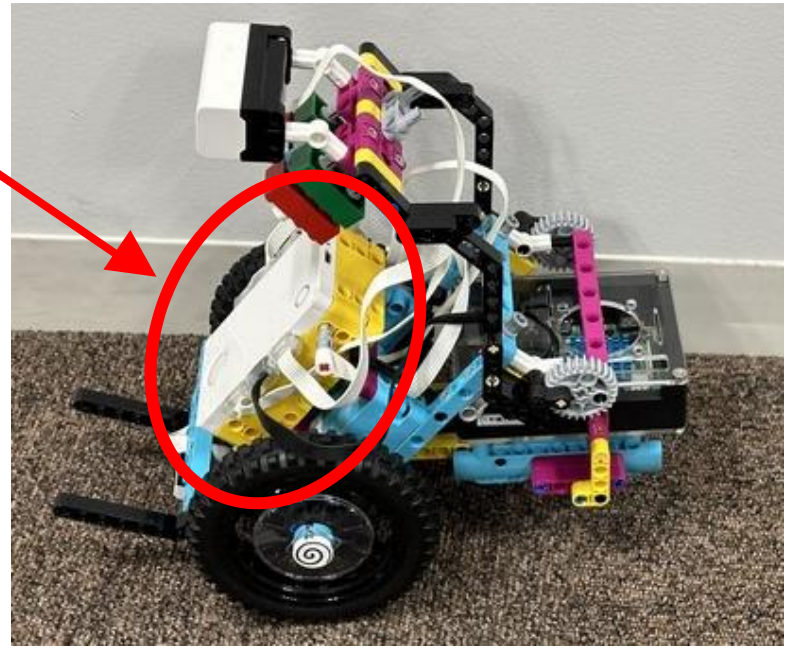



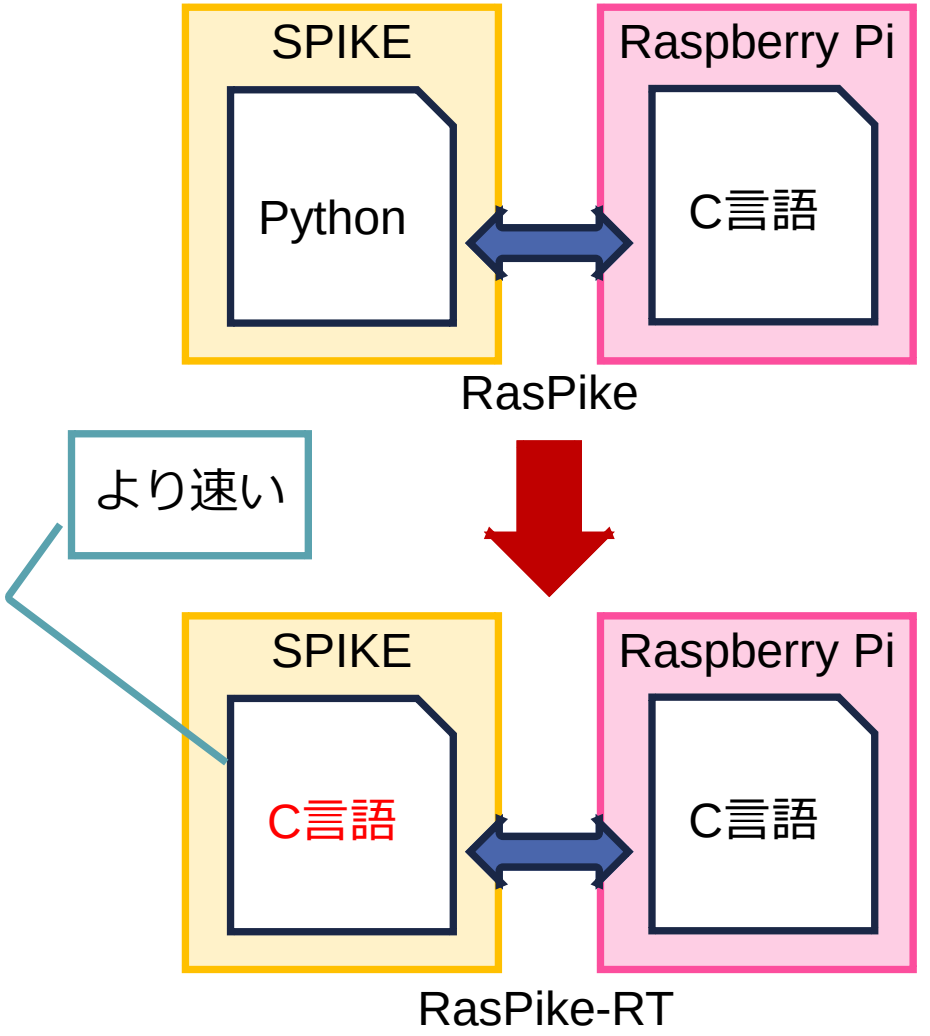
# エージェントプログラムをSPIKE-RT化

- RasPikeのエージェントプログラム(SPIKE側プログラム)のみをSPIKE-RT化
  - RasPike-RTと呼ぶ
  - ソフトウェア構成はRasPikeと同じ
  - RasPikeはMicroPythonで記述されていた
  - SPIKE-RT化する事でレスポンスが向上




SPIKE-RT
プログラム



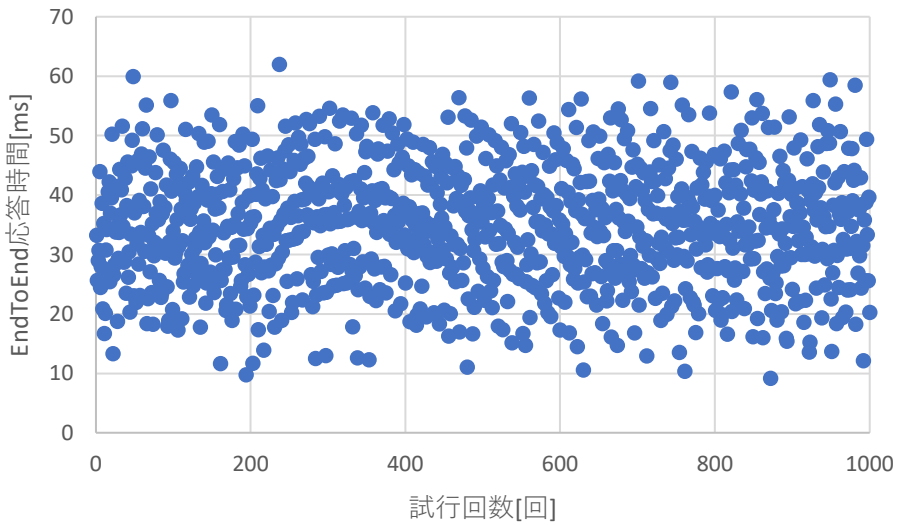


# 参考：RasPike-RTの応答時間

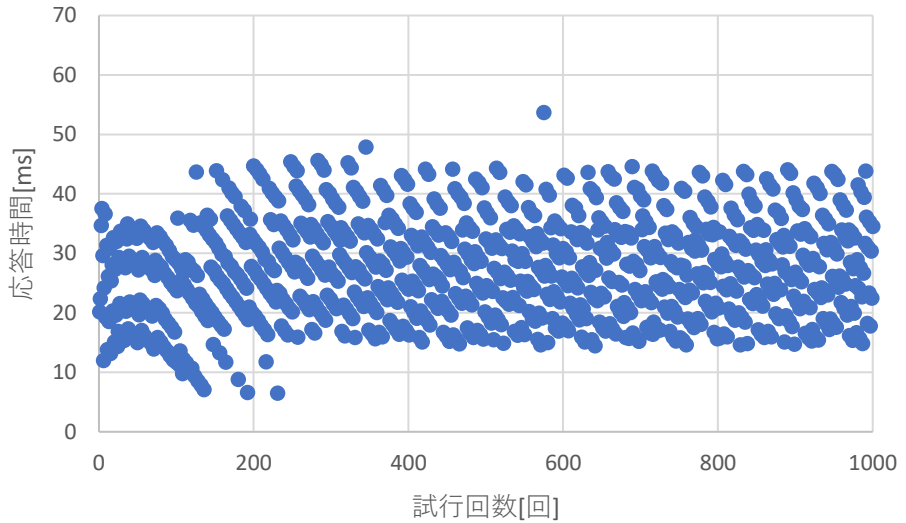
- 結果
  - 最悪値 (最大値)
    - RasPike：61.9ms
    - RasPike-RT：53.7ms
  - 平均値
    - RasPike：34.7ms
    - RasPike-RT：27.3ms

- カラーセンサが黒を検知したらモータを停止するプログラムを実行
- 黒を検知してから，実際にモータが停止するまでの時間を計測

応答時間




RasPike




RasPike-RT

# まとめ

- RasPike
  - 公式にサポートされている
  - ETロボコンでの動作実績あり

- SPIKE-RT
  - 高いレスポンス
  - ネットワークが不要
  - 組込みソフトの開発技術を学べる

- RasPike-ROS
  - Pythonでプログラミング可能
  - ROS 2に触れられる
  - ローカルPCで開発が可能

- RasPike-RT
  - RasPikeから大きく変更せずにレスポンスを向上
  - Hubに書き込んでしまえば何も変わらない

SPIKE-RT

RasPike-ROS

RasPike-RT
(エージェントプログラム)

# 目次

- 自己紹介
- RasPike・SPIKE-RT・RasPike-ROSの位置付けを整理
- SPIKE-RTの環境構築方法
- ROS 2とRasPike-ROSの紹介

# SPIKE-RTの環境構築

- 環境構築方法をまとめたWikiと，それに従って作業をした動画を作成
  - Wiki：https://github.com/Hiyama1026/spike-rt-sample/wiki
  - WSL2またはUbuntu22.04（ネイティブ）を想定

- WSL2環境にSPIKE-RTをインストールする動画をこのあと紹介

**SPIKE-RT環境構築Wiki**

# 環境構築

- ドライバ等のインストールからシリアル通信サンプルの実行までの手順
  - https://www.youtube.com/watch?v=XJ7rltuf3Jc

# 目次

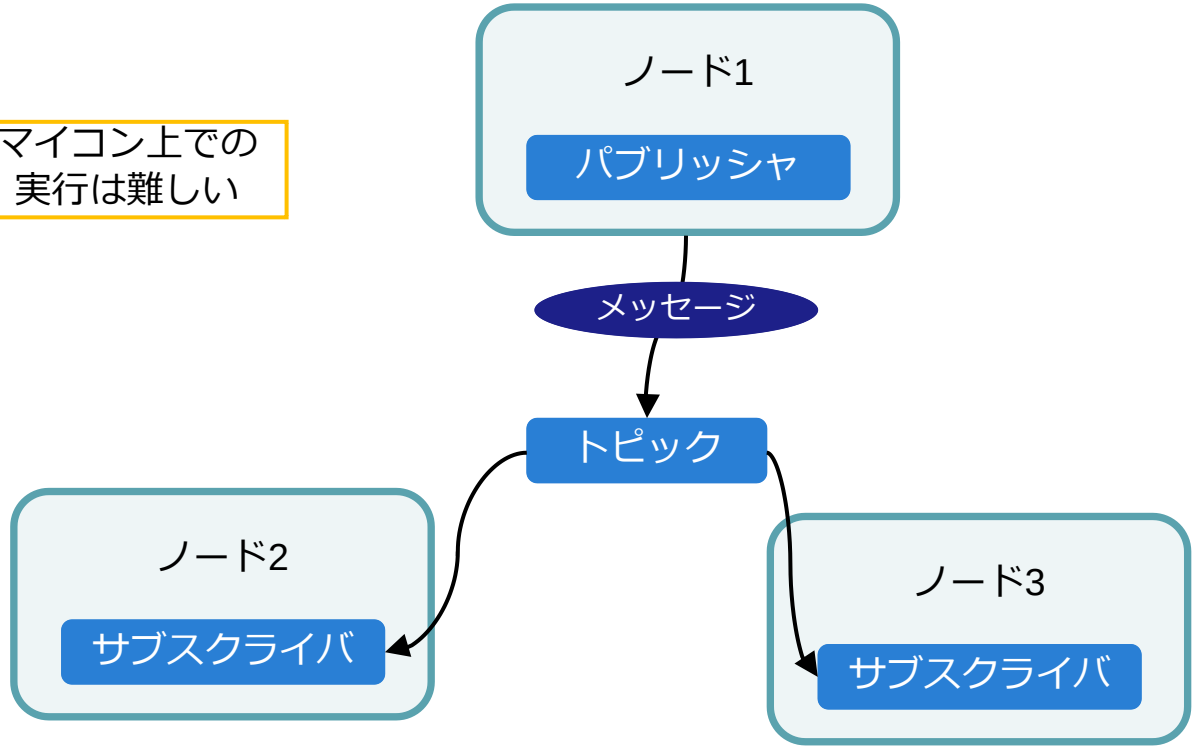
- 自己紹介
- RasPike・SPIKE-RT・RasPike-ROSの位置付けを整理
- SPIKE-RTの環境構築方法
- ROS 2とRasPike-ROSの紹介

# ROS 2

- ROS 2とは
  - Robot Operating Systemの略
  - ロボットや自動運転向けの分散処理フレームワーク
    - 通信ミドルウェア
    - publish/subscribe通信が可能
  - UNIX系OS上での実行を想定

マイコン上での
実行は難しい

- 活用例
  - Amazon Roboticsの物流補助ロボット
  - aibo（SONY）

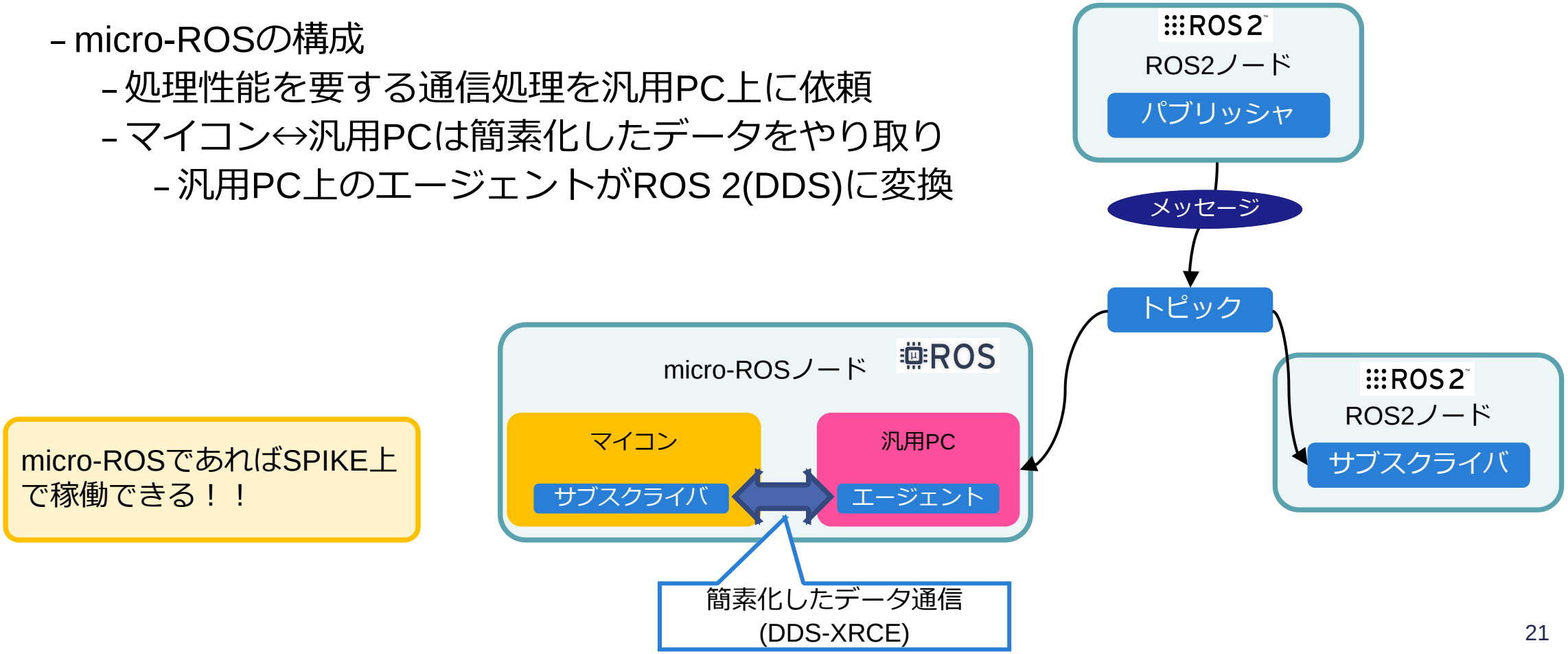



- SPIKE Prime Hubもマイコン
- ROS2の稼働は難しい

# micro-ROS

- ROS 2との関係性
  - ROS 2のマイコン上での実行は厳しい
  - マイコンをROS2に接続する為の機構

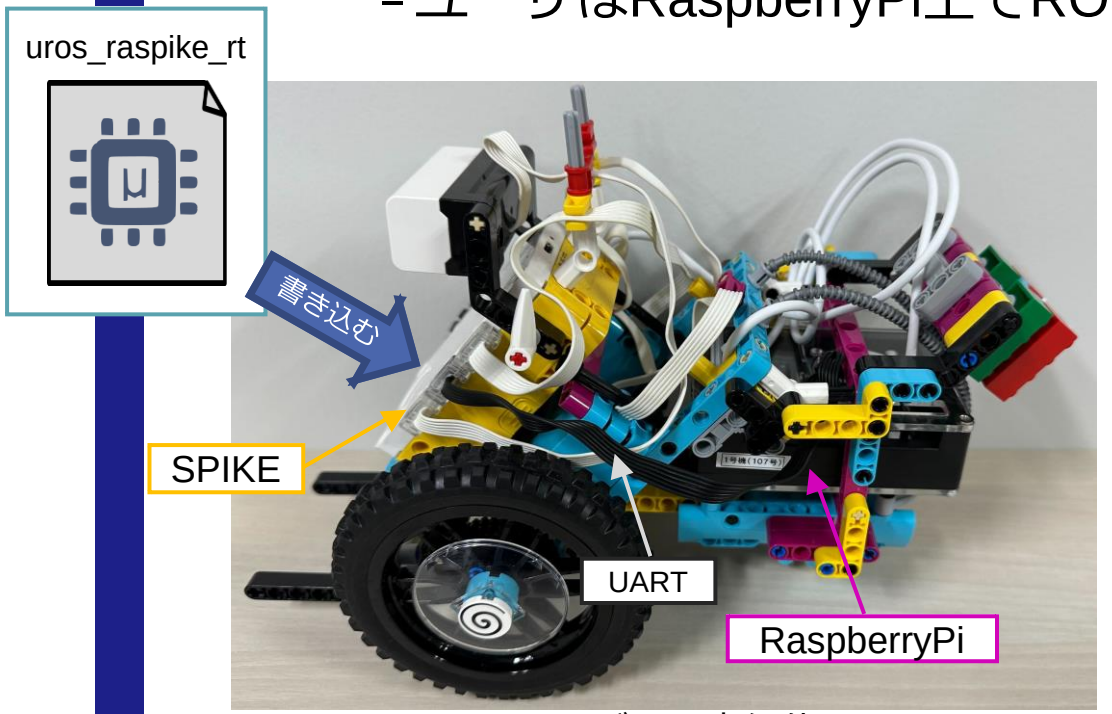
- micro-ROSの構成
  - 処理性能を要する通信処理を汎用PC上に依頼
  - マイコン↔汎用PCは簡素化したデータをやり取り
    - 汎用PC上のエージェントがROS 2(DDS)に変換



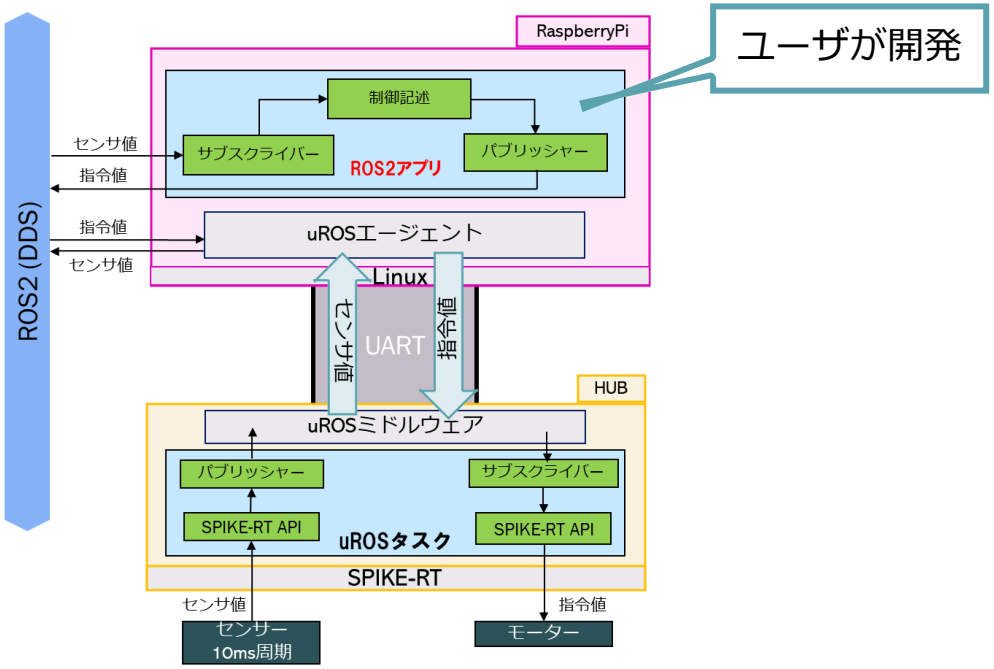
# ETロボコン走行体をROS 2で動かす

- ETロボコン走行体
  - SPIKEとRaspberryPiを使用
- RasPike-ROS
  - ETロボコン走行体向けソフトウェアプラットホーム
  - ROS2とmicro-ROSを使用
  - SPIKEにmicro-ROSファームウェアを書き込む
  - ユーザはRaspberryPi上でROS2アプリケーションの開発を行う

RasPike-ROS




ETロボコン走行体



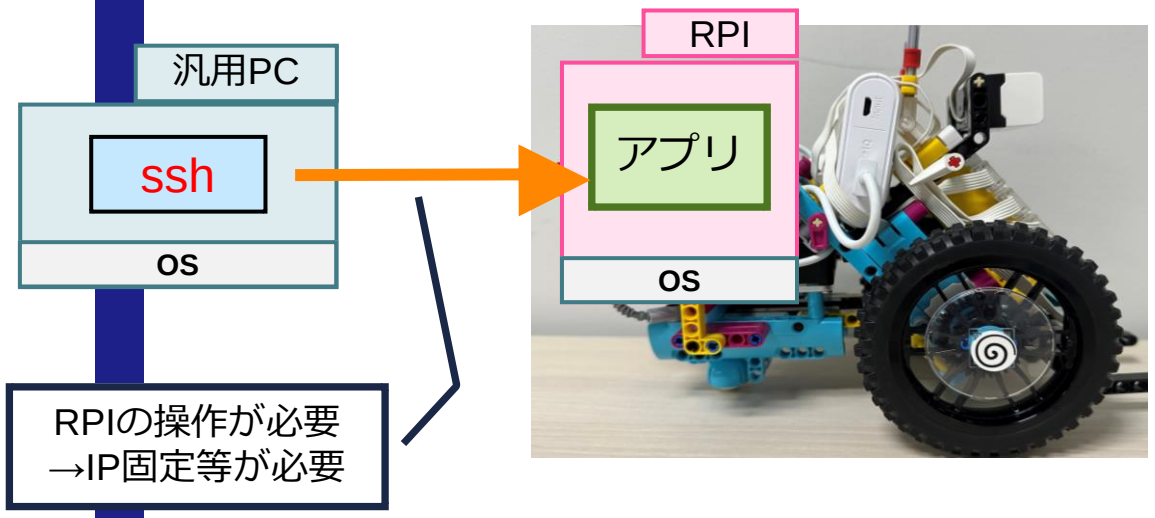

RasPike-ROSの内部構成

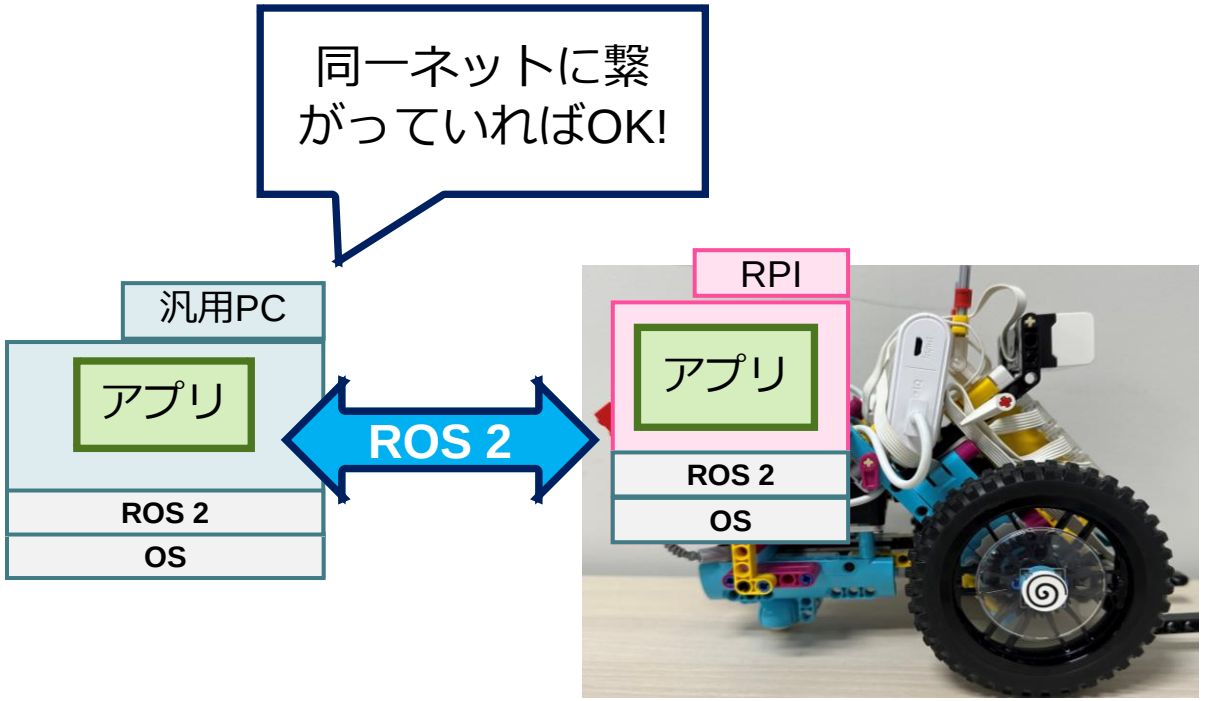
# RasPike-ROSの主な特徴

「2022年のROSダウンロード数」
4907万8176件(※1)

※1：http://wiki.ros.org/Metrics

- Pythonで開発
- ROSはロボット制御の分野において広く使われる技術
  - ソフトウェア資産が蓄積されている
  - カメラ，コントローラー等をROSで接続可能
- 外部PCでアプリの開発・実行が可能
  - 自分のPC上で開発可能
  - RaspberryPi操作が不要



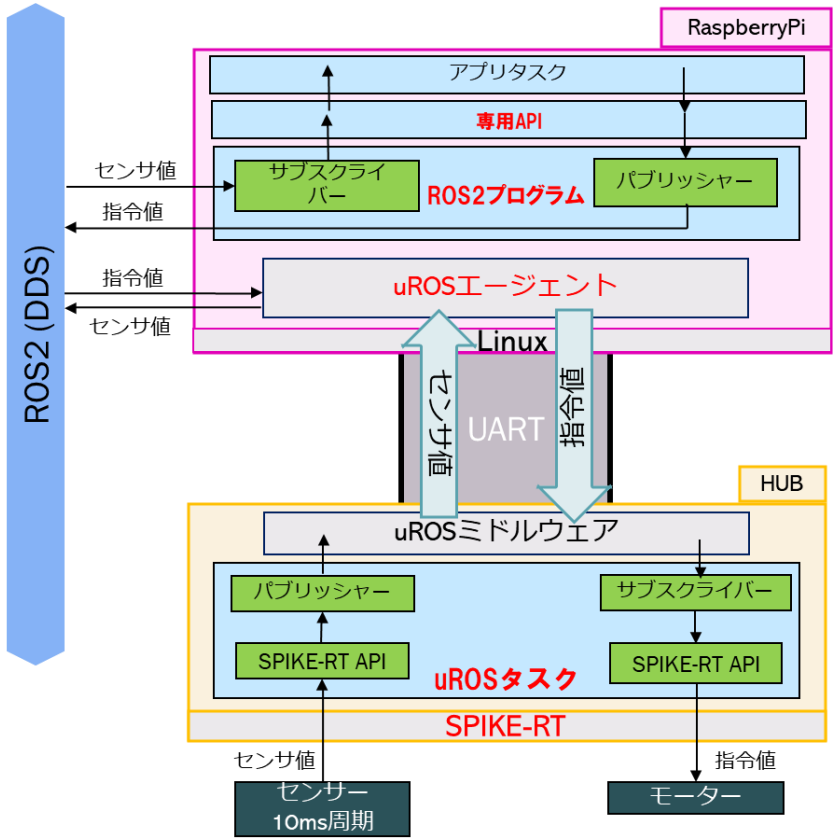



# RasPike-ROS専用APIを用意

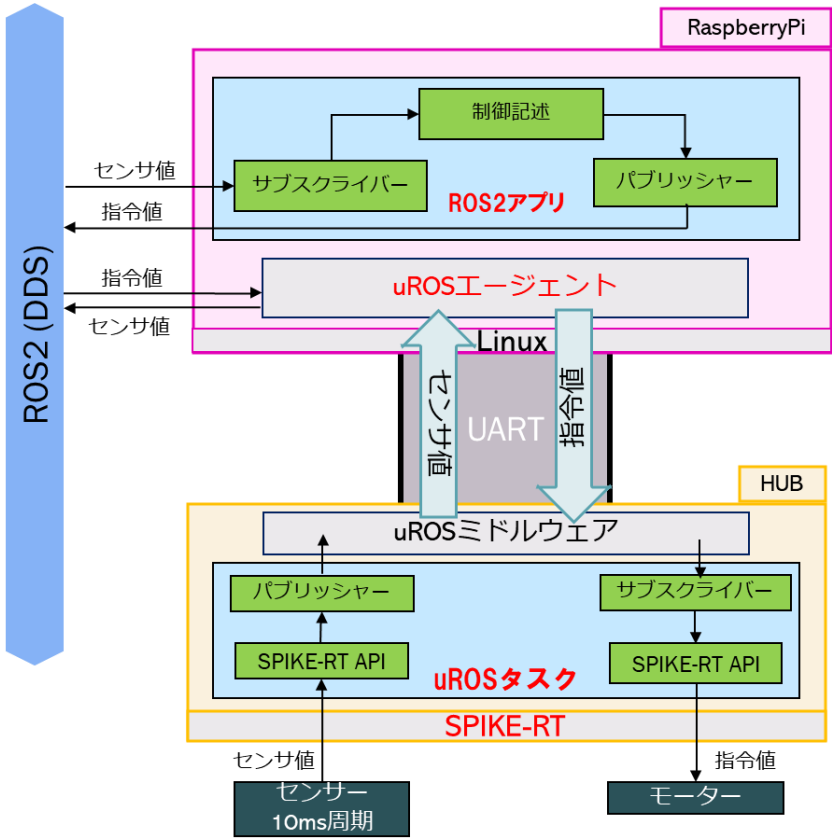
ROS2の知識がなくても開発が可能（Python）

ROSの学びになる．ROS2ソフトウェア資産の再利用が可能．

- RasPike-ROSでのアプリ開発専用のAPIを用意
  - 2種類の方法でアプリ開発が可能（記述例は後述）
    - アプリ開発専用APIを用いる方法
    - ROS2 APIを直接扱う方法




専用APIを使用した場合




ROS2APIを使用した場合

# ETロボコン走行体をROS 2で動かす

- ライントレースプログラム

**Python**

```python
def line_trace_on_tick(self):
    self.steering_amount_calculation()
    self.motor_drive_control()

def timer_on_tick(self):
    # メッセージの生成
    motor_speed = MotorSpeedMessage()
    color_mode = Int8()
    color_mode.data = 3
    motor_speed.right_motor_speed = self.send_right_speed
    motor_speed.left_motor_speed = self.send_left_speed

# メッセージのパブリッシュ（送信）
    self.motor_speed_publisher.publish(motor_speed)
    self.color_mode_publisher.publish(color_mode)

def main(args=None):
    # ROS通信の初期化
    rclpy.init(args=args)
    # ノードの生成
    node = linetracerNode()

    rclpy.spin(node)      # ROS2アプリ始動
    node.destroy_node()
    rclpy.shutdown()
```

タイマーコールバック(周期呼び出し)

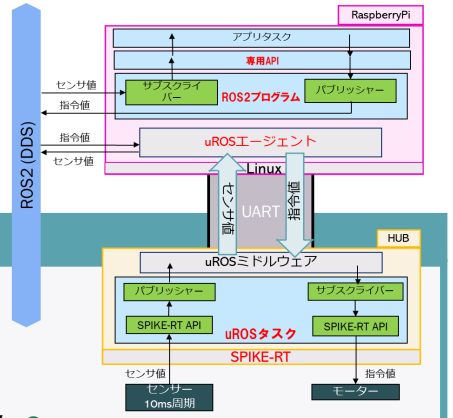
ROS2 API（パブリッシュ）

**Python**

```python
# ライントレース
def steering_amount_calculation(self):
    target_brightness = (white_brightness - brack_brightness) / 2
    diff_brightness = target_brightness - self.rev_color_sensor_refrection
    self.diff[1] = self.diff[0]
    self.diff[0] = int(diff_brightness)

    p_val = diff_brightness
    i_val = self.pre_i_val + (self.diff[0] + self.diff[1]) * delta_t / 2
    d_val = int((self.diff[0] - self.diff[1]) / 1)

self.steering_amount = (kp * p_val) + (kd * d_val) + (ki * i_val)

def motor_drive_control(self):
    self.send_left_speed = bace_speed + (self.steering_amount * left_edge)
    self.send_right_speed = bace_speed - (self.steering_amount * left_edge)
```

# RasPike-ROS専用APIを用いたアプリの記述例

- 独自のAPIを用意（オブジェクト指向のAPI）
- 周期呼び出しされる関数にユーザーはコードを記述する




周期呼び出し

専用API

**Python**

```python
# timer callback
def app_timer(self):
    # カラーセンサのモード指定
    color_sensor.set_color_mode(4)
    # カラーセンサ値の取得
    self.rgb_val = color_sensor.get_rgb()
    # モードが切り替わっているかを確認
    if self.rgb_val[2] < 0:
        return
    self.steering_amount_calculation()
    self.motor_drive_control()


# メイン
def main(args=None):
    # ros2 start
    rpi_ros2_node_start(args)

if __name__ == "__main__":
    main()
```

main関数・周期関数

**Python**

```python
# ライントレース
def steering_amount_calculation(self):
    target_brightness = (white_brightness + brack_brightness) / 2
    diff_brightness = target_brightness - self.rgb_val[2]
    self.diff[1] = self.diff[0]
    self.diff[0] = int(diff_brightness)

    p_val = diff_brightness
    i_val = self.pre_i_val + (self.diff[0] + self.diff[1]) * delta_t / 2
    d_val = int((self.diff[0] - self.diff[1]) / delta_t)

    self.steering_amount = (kp * p_val) + (kd * d_val) + (ki * i_val)

def motor_drive_control(self):
    motor.set_left_motor_speed(int(bace_speed + (self.steering_amount * left_edge)))
    motor.set_right_motor_speed(int(bace_speed - (self.steering_amount * left_edge)))
```

専用API

ライントレース演算

# SPIKE-RTでのロボコンアプリ記述例

- タスク記述はRasPikeと同等
  - ITRON系
- モータ等のAPIが異なる


**C言語**

```c
void
trace_task(intptr_t exinf) { //周期タスク
  /*カラーセンサー値の取得*/
  reflection =  pup_color_sensor_reflection(col);

  /*操舵量の計算・走行モーター制御*/
  steering_amount_calculation();
  motor_drive_control();
}


void
main_task(intptr_t exinf) {
  ＜省略（デバイスセットアップ等）＞

  // trace_task()用タイマ
  sta_cyc(TRACE_CYC);

  while (1){
    slp_tsk();
  }
}
```

SPIKE-RT API

main関数・周期タスク

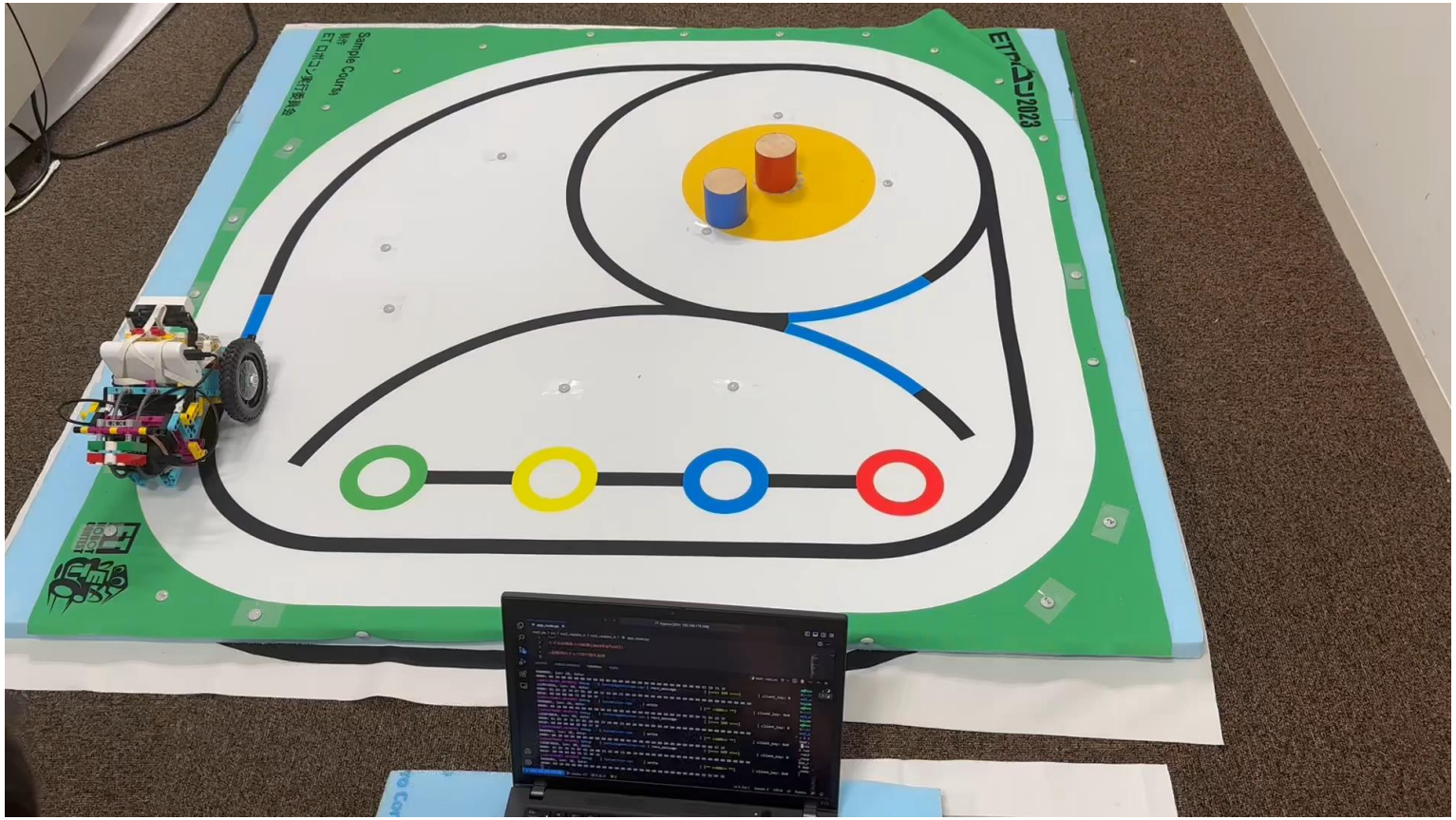
**C言語**

```c
static void motor_drive_control(void){
  int left_motor_powor, right_motor_powor;

  /*走行エッジを右にする場合はRIGHT_EDGEに切り替える*/
  left_motor_powor = (int)(BACE_SPEED + (steering_amount * LEFT_EDGE));
  right_motor_powor = (int)(BACE_SPEED - (steering_amount * LEFT_EDGE));

  /*左右モーター駆動パワーの設定*/
  pup_motor_set_power(l_motor, left_motor_powor);
  pup_motor_set_power(r_motor, right_motor_powor);
}

static void steering_amount_calculation(void){
  /*目標輝度とカラーセンサー値の差分を計算*/
  diff[0] = diff[1];
  diff[1] = (float)(target_brightness - reflection);
  /*ステアリング操舵量を計算*/
  integral += (diff[1] + diff[0]) / 2.0 * DELTA_T;
  p = KP * diff[1];
  i = KI * integral;
  d = KD * (diff[1] - diff[0]) / DELTA_T;
  steering_amount = p + i + d;
}
```

SPIKE-RT API

ライントレース演算・出力

# ETロボコン走行体をROS 2で動かす




動画：https://www.youtube.com/watch?v=RoaVhumuqcQ