ROSCon JP 2021
2021/09/16

ROS 2 Client Library for E^2

高瀬 英希 (東京大学/JSTさきがけ)
武田 大輝・今西 洋偉・
祐源 英俊 (京都大学)

自己紹介

@takasehideki

本務・兼務

- 東京大学 情報理工学系研究科 准教授
- JSTさきがけ 兼任研究者
- 一般社団法人ROSCon JP 理事
- TOPPERSプロジェクト 運営委員

学会活動

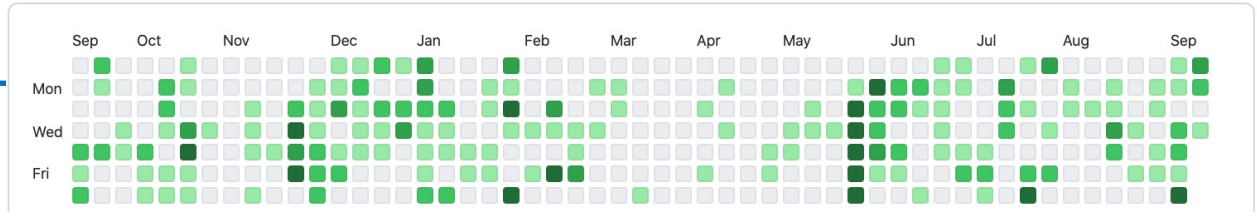
- IPSJ-SIGEMB 運営幹事
- IEICE-RECONF 専門委員
- SWEST ステアリング委員
- ROSCon JP 実行委員

コミュニティ活動

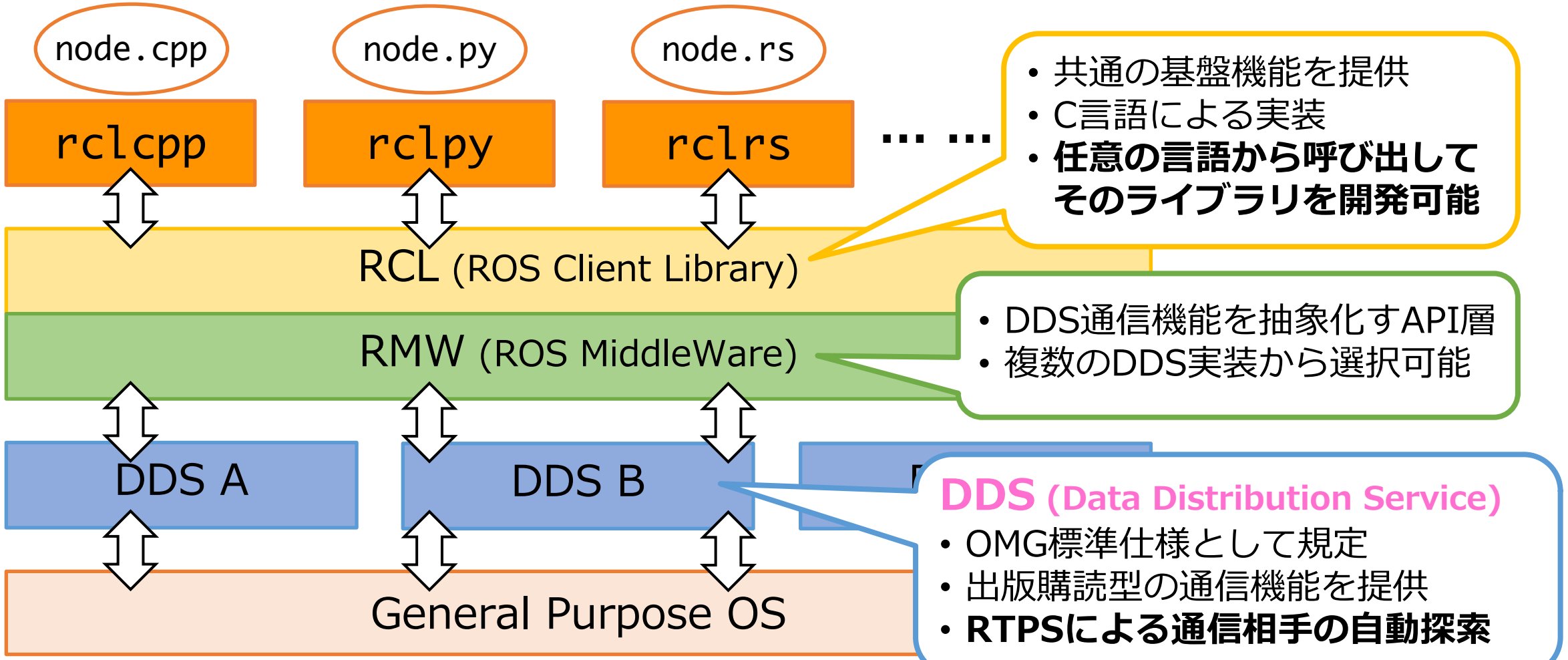
- ROS Japan Users Group
- IoT ALGYAN (あるじゃん) 運営委員
- NervesJP：Elixir for IoT

1,220 contributions in the last year

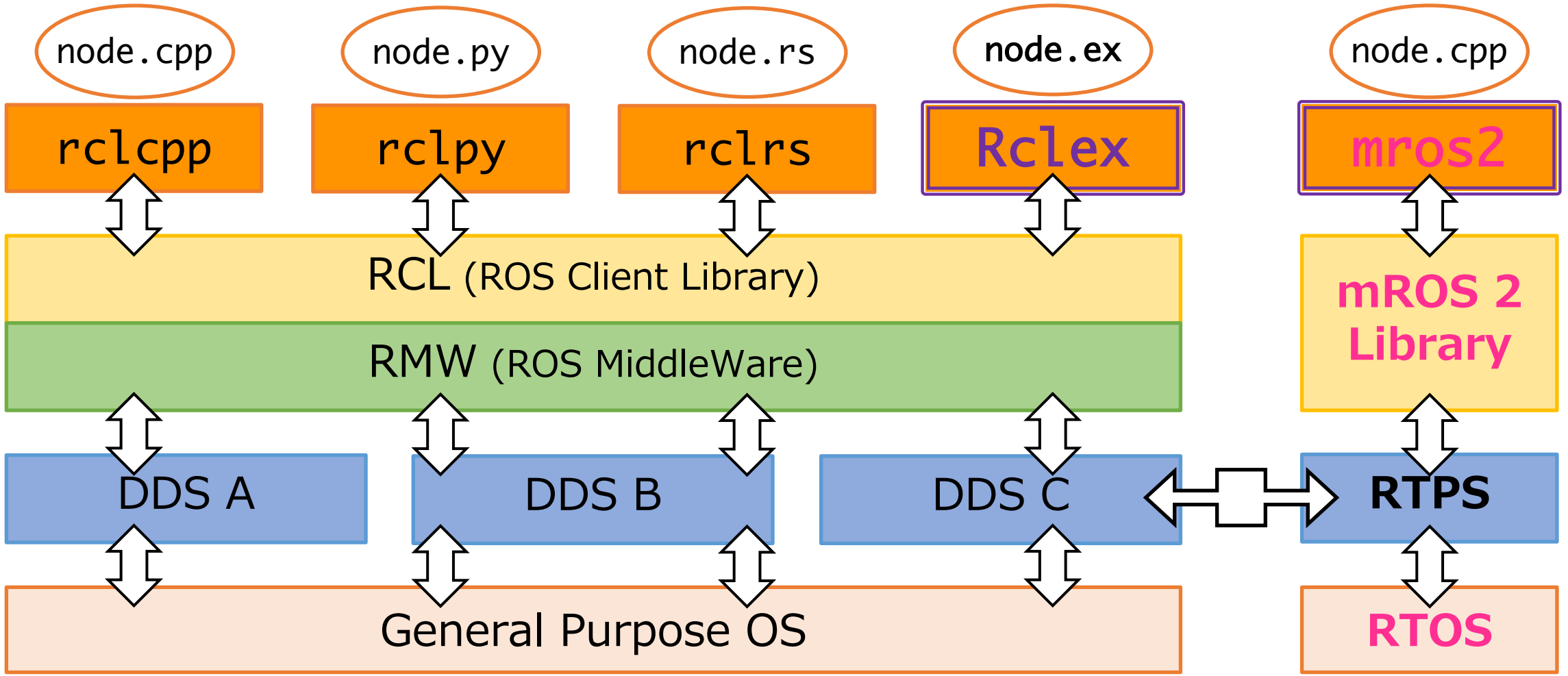
Contribution settings



Prologue: ROS 2 の階層構造



Client Library for„



ROS 2 Client Library for Elixir

Elixir とは？

elixir

2012年に登場した新しめの関数型言語

Erlang VM上で動作
- 高い並行／並列性能を誇る
- 軽量かつ頑強なプロセスモデル
- 耐障害性が極めて高い

Rubyを基にした言語設計
- 習得しやすく生産性が向上する
- レスポンス性能が極めて高い
- **Web/IoT/AIライブラリを備える**

Elixir *Zen* Style

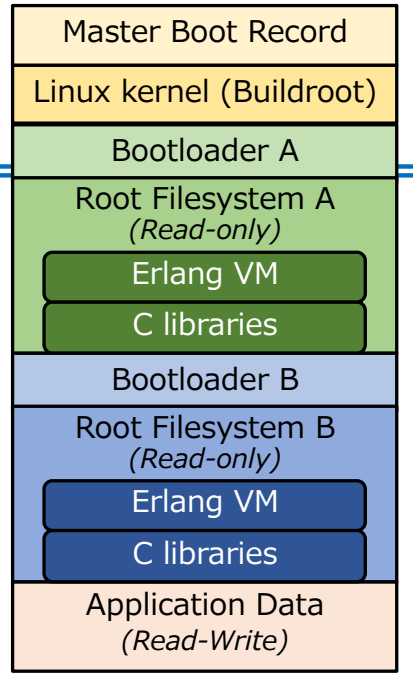
```
1..1000
  |> Flow.from_enumerable()
  |> Flow.map(& foo(&1))
  |> Flow.map(& bar(&1))
  |> Enum.to_list
  |> Enum.sort
```

Programming should be about transforming data

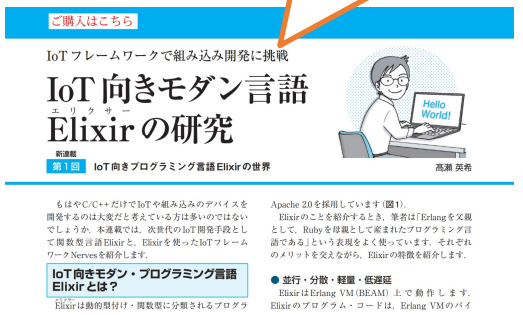
- データフローと並列処理を **Enum Flow** **|>** で直感的に記述できる

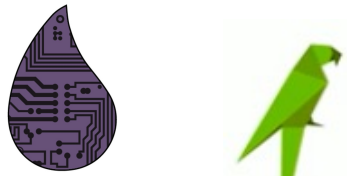
NERVES

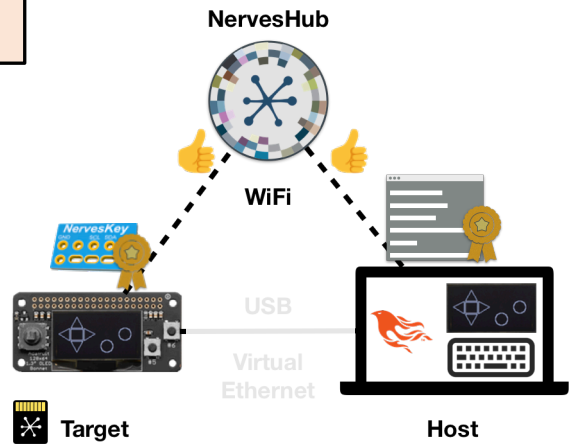
- **ナウでヤングなcoolなすごいやつ!!**
  - –極小規模のファームウェア (~>30 MB)
  - –堅牢性の高いファイルシステム
  - –機能の取捨選別が容易 (Buildrootの利用)
- Elixirと完全に互換な言語機能
  - –アクターモデルで並行処理ができる
  - –IoT向けになにか気にする必要がない
  - –有用なライブラリとツールが揃っている
- ➢日本初!?のNerves搭載製品
  → Pocket LANcher



CQ出版Interface
2020年6月号～
2021年1月号





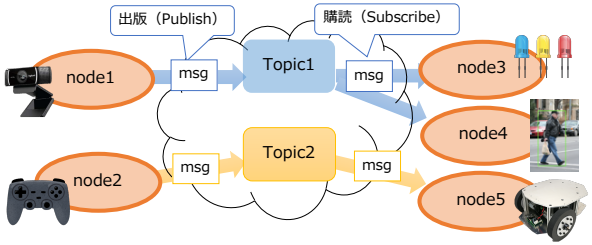


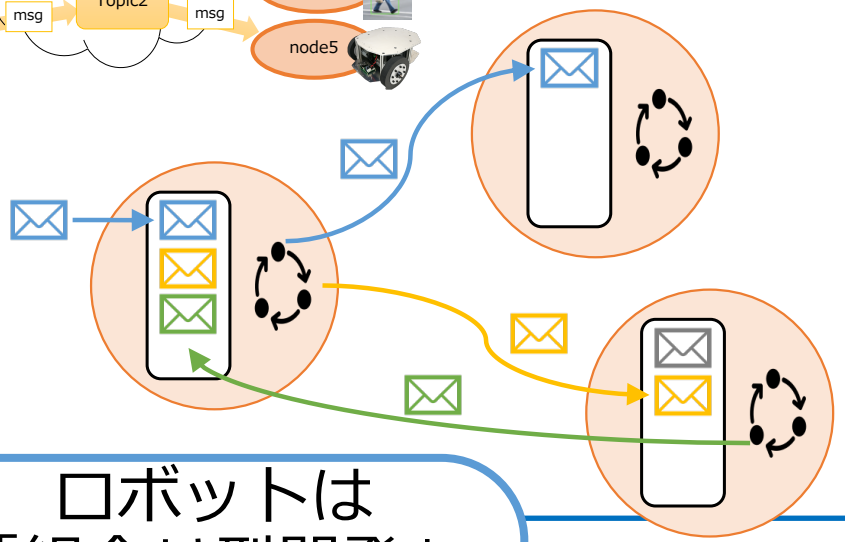
NERVES JP
Nerves Community in Japan

ROS開発になぜ Elixir なのか？

ロボットプログラミングは「関数型」だ！！

ROSの本質は
通信にあり





Phoenix Framework

ERLANG

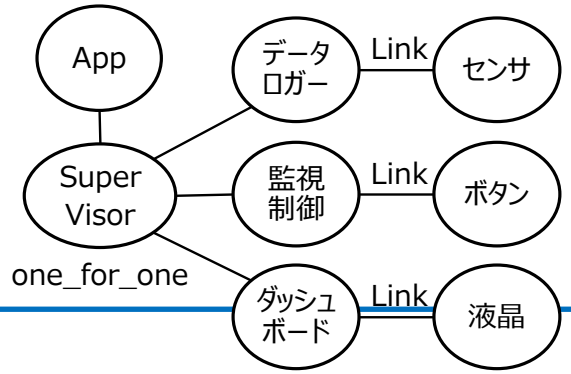
NERVES

Nx

デバイス制御と
知的処理がキモ

```
{:ok, ref} = I2C.open(@i2c_bus)
I2C.write(ref, @i2c_addr, <<0xBE, 0x08, 0x00>>)
I2C.write(ref, @i2c_addr, <<0xAC, 0x33, 0x00>>)
{:ok, val} = I2C.read(ref, @i2c_addr, 7)
<<_state::8, raw_humi::20,
    raw_temp::20, _crc::8>> = val
```

Elixir Circuits



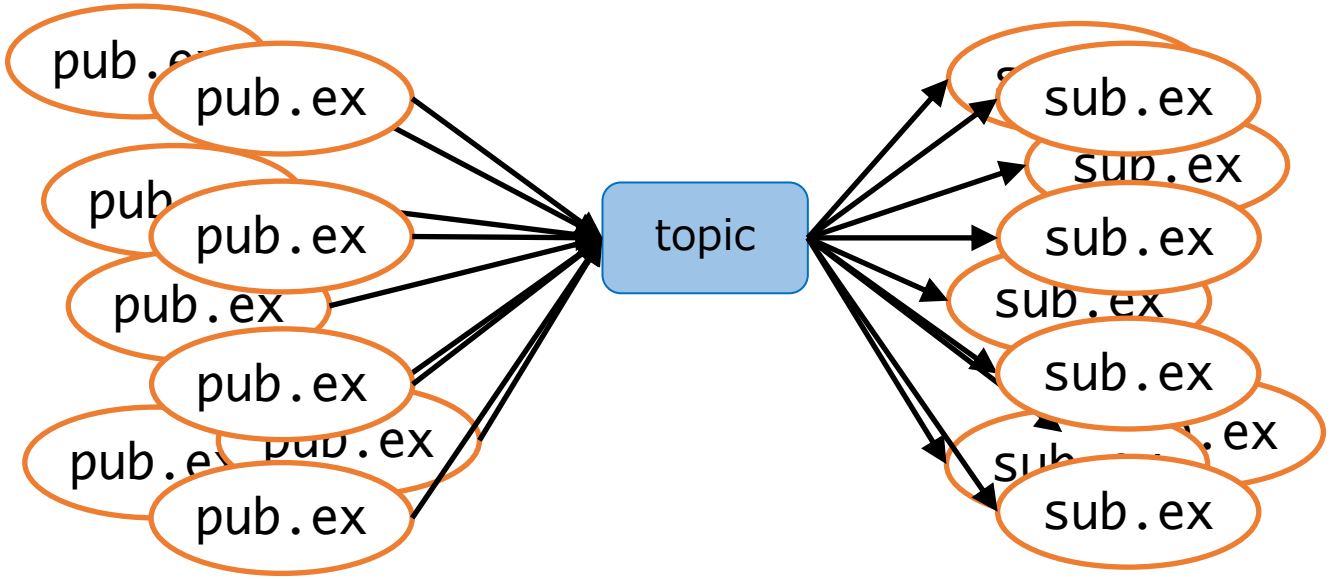
ロボットは
「組合せ型開発」

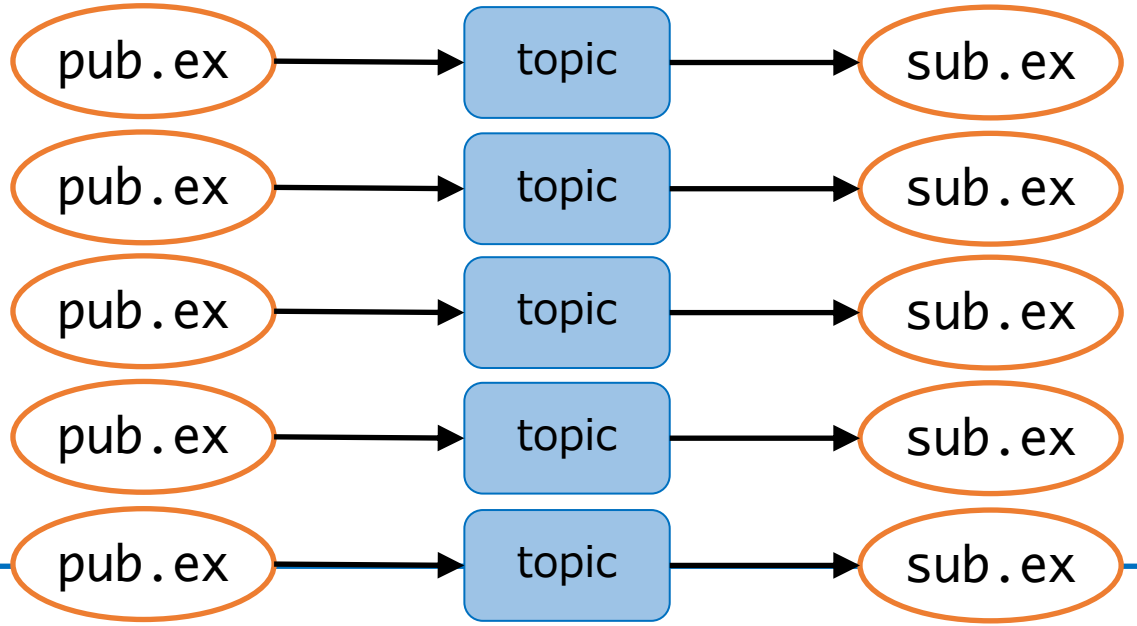
壊れてなんぼ
でも壊れたら困る

Rclexの強みと実現できること

軽量プロセスの並行処理によって
ROS 2の通信スケーラビリティを向上させる

- ElixirアプリからErlangプロセスを同時に大量生成させる
- 出版購読の通信処理とコールバックを非同期に実行する





github.com/rclex

Rclex によるプログラミング例

```
1   defmodule RclexNode do
2     def pub(num_node) do
3       Rclex.rclexinit()
4       |> Rclex.create_nodes('rclex_node', num_node)
5       |> Rclex.create_publishers('chatter', :single)
6       |> Rclex.Timer.timer_start(1000, &func/1)
7     end
8
9     def func(publisher_list) do
10      n = length(publisher_list)
11      msg_list = Rclex.initialize_msgs(n, :string)
12      Enum.map(0..(n - 1), fn index ->
13        data = "Hello World from rclex_node_" <> to_string(index)
14        IO.puts("publish message: #{data}")
15        Rclex.setdata(Enum.at(msg_list, index), data, :string)
16      end)
17
18      Rclex.Publisher.publish(publisher_list, msg_list)
19    end
20  end
```

インデックス付きの
名前でノードを生成

ノード数

パイプ演算子で
ノード設定の処理

トピックの作成
:single->1つのトピック
:multi-> 複数のトピック

callback関数と
タイマ周期の設定

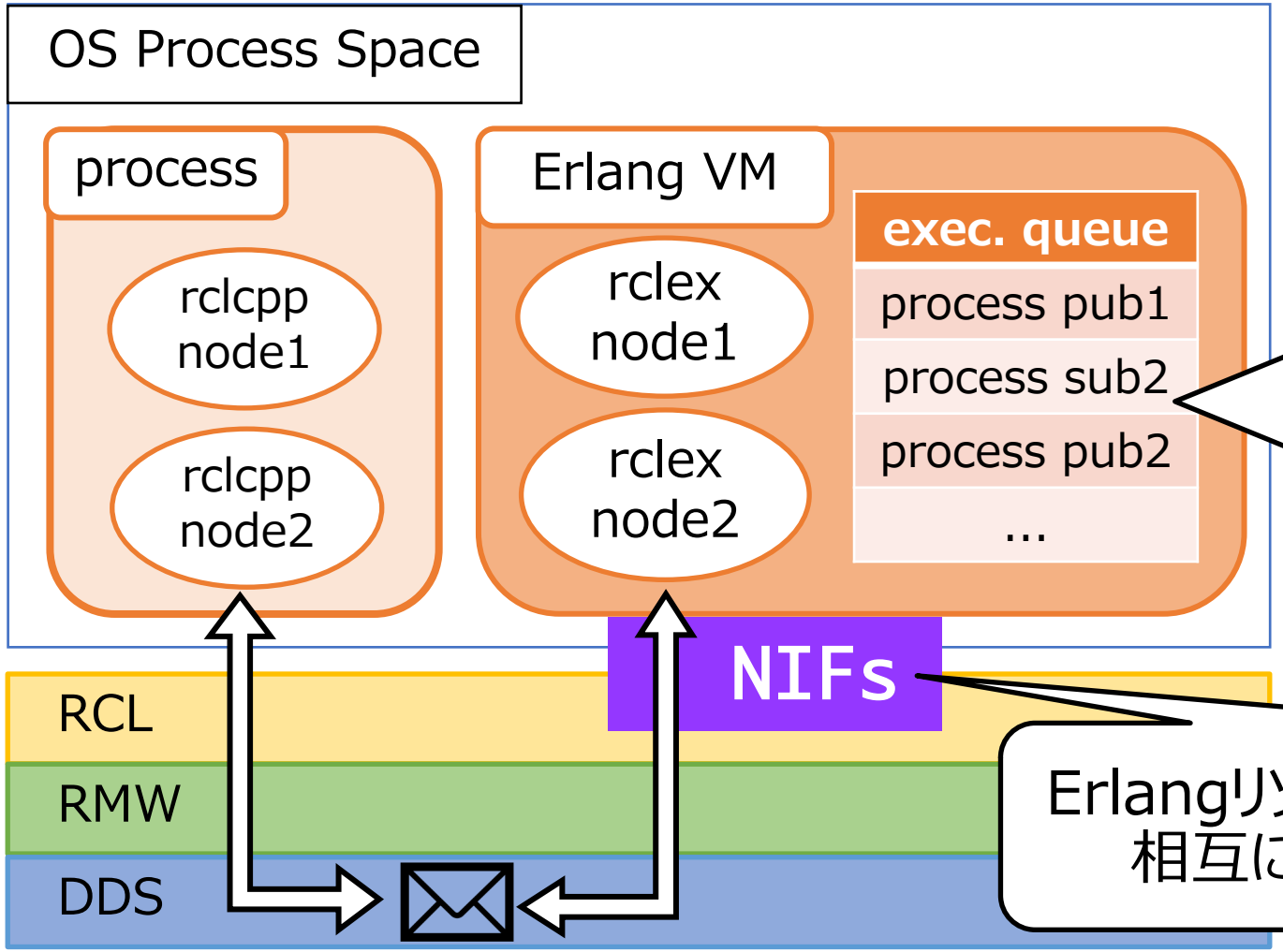
Enumによる
データ加工

ノードごとの
メッセージを
リストで用意

データ出版

github.c

Rclexの内部実装

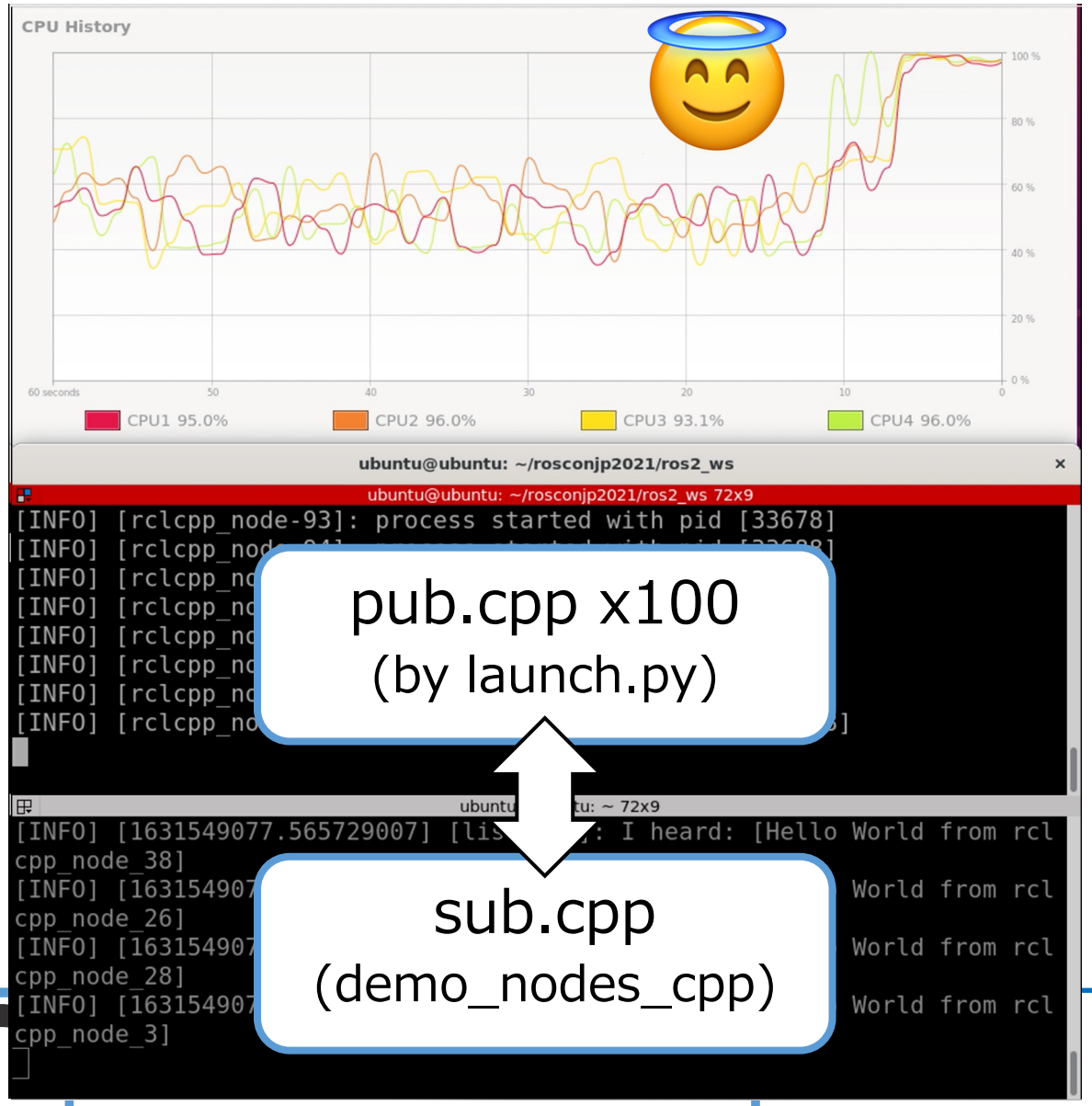


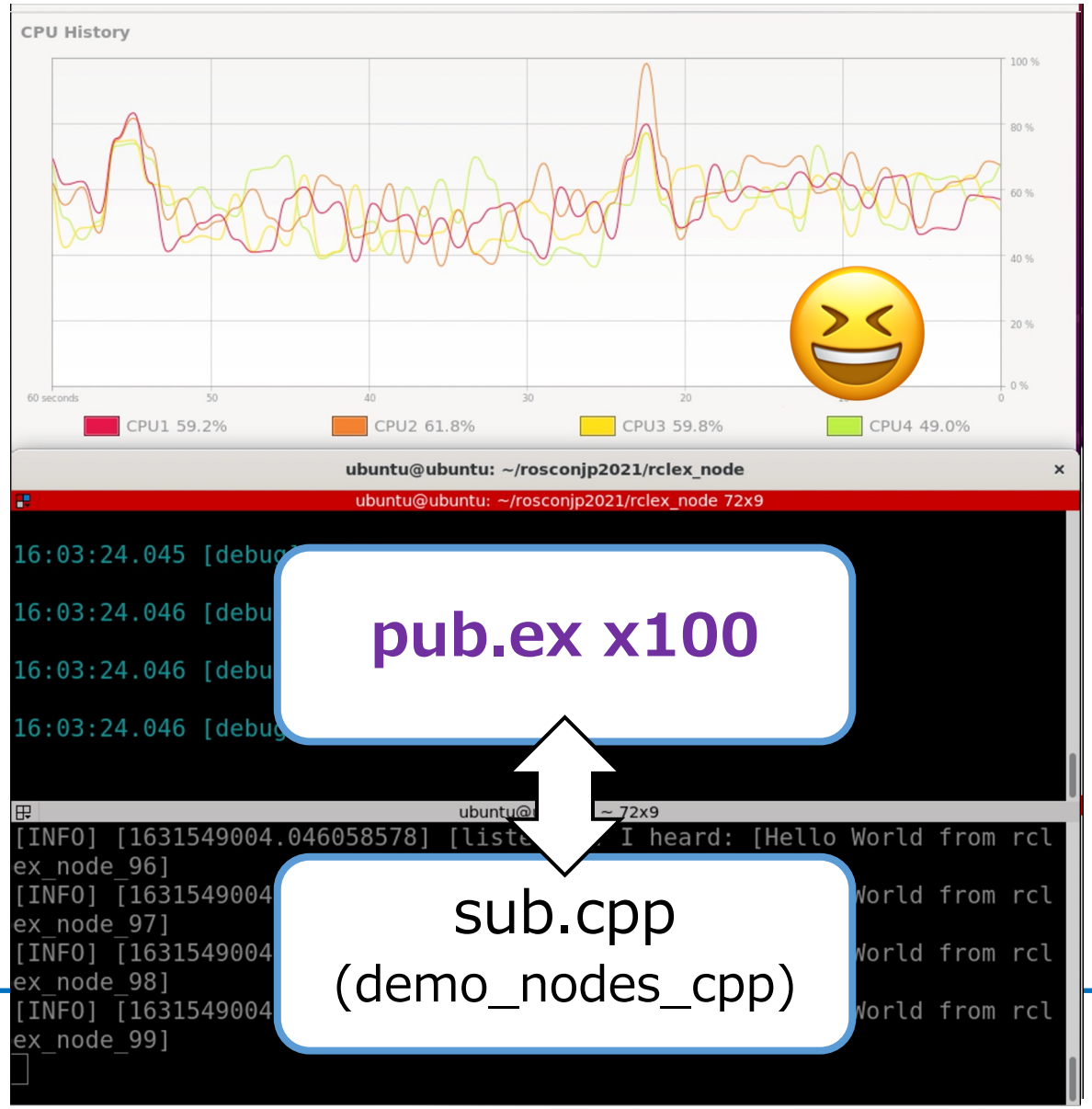
- Erlang VMはOSプロセスとして駆動
- スケジューラでElixirプロセスを実行
  = Rclexにおける pub/sub ノード
- **プロセスが軽量**
  - ✓ 起動時間：数マイクロ秒
  - ✓ メモリ：およそ300ワード

ErlangリソースオブジェクトとRCLの構造体を相互に変換してRCL APIを呼び出す

論よりRUN!! on RPi4/Foxy

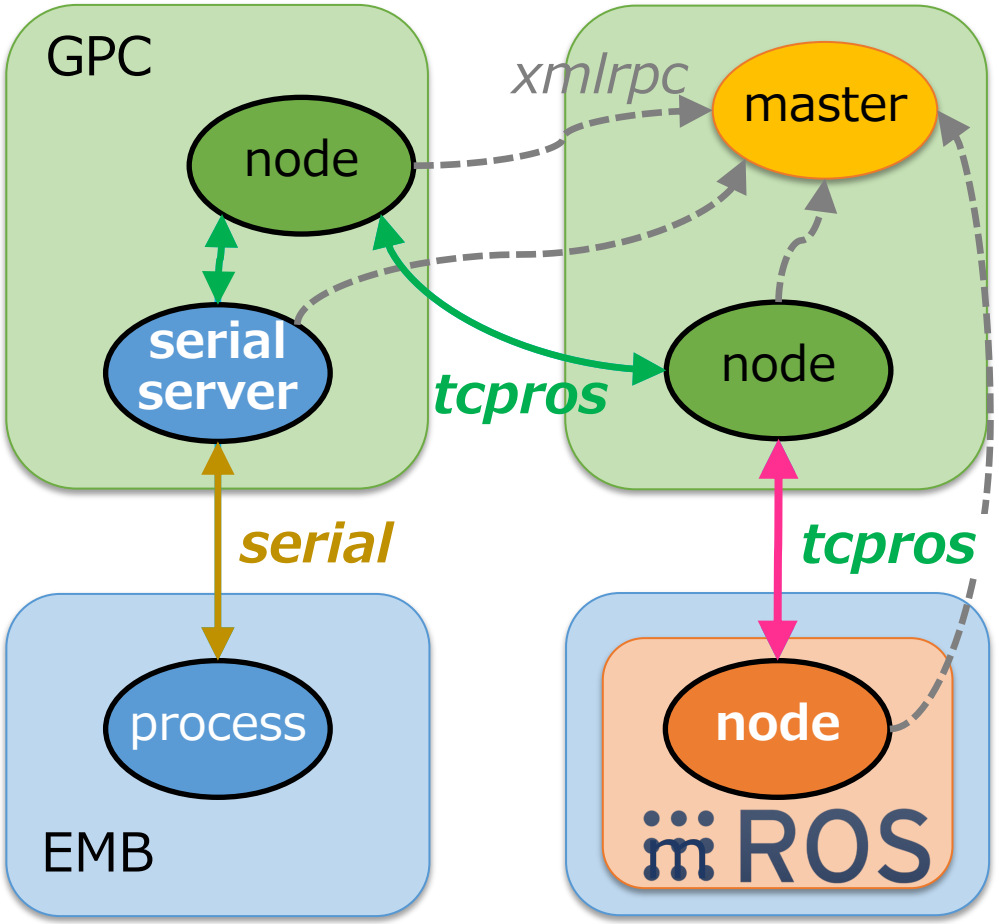
rclex/rcjp21_demo



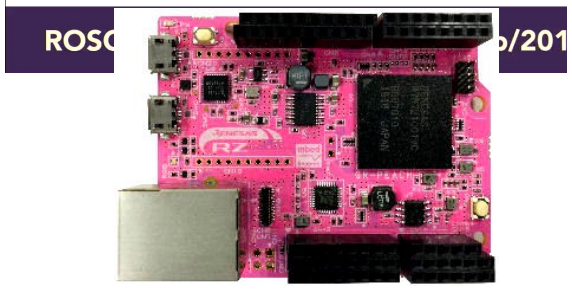


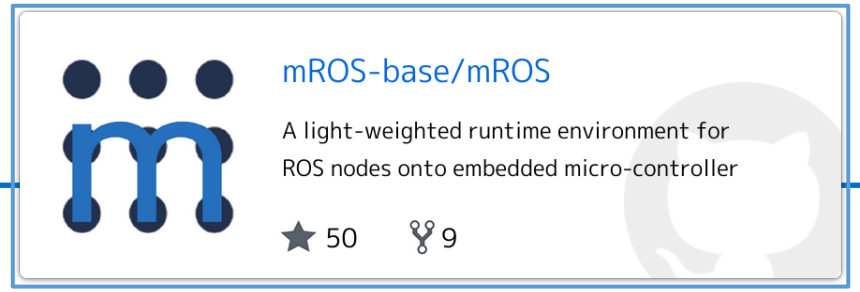
agent-less and *lightweight*
runtime environment
for embedded devices

昔話：ROS 1の通信方式と組込み対応

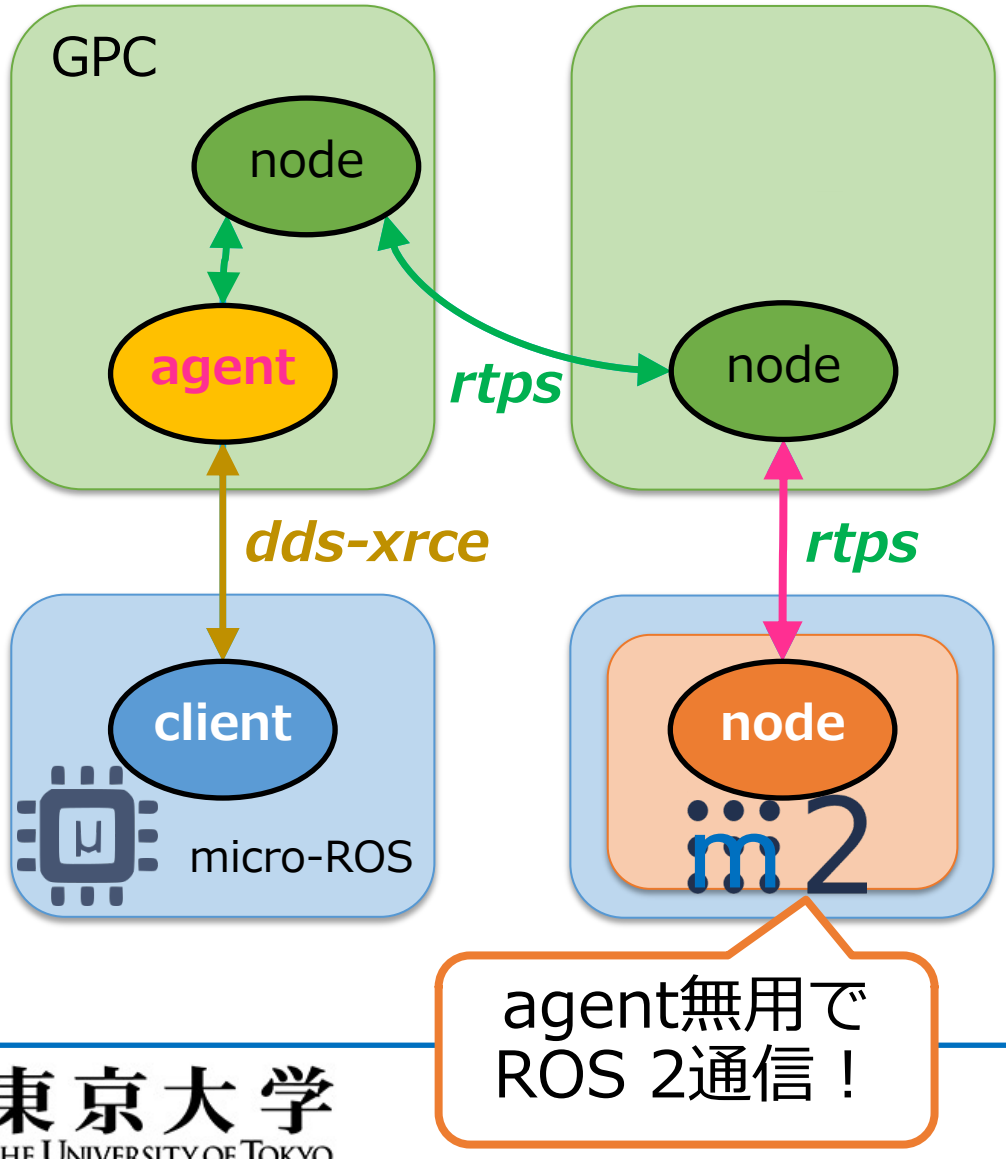


※GPC: General Purpose Computer
EMB: Embedded micro-computer



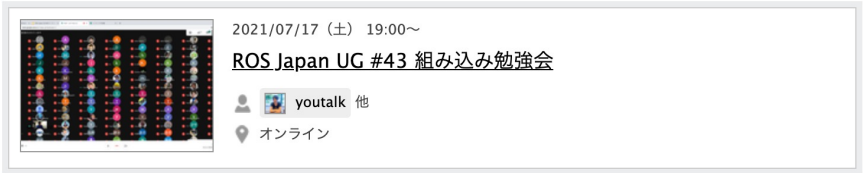


そして時代は ⁝2 に!!



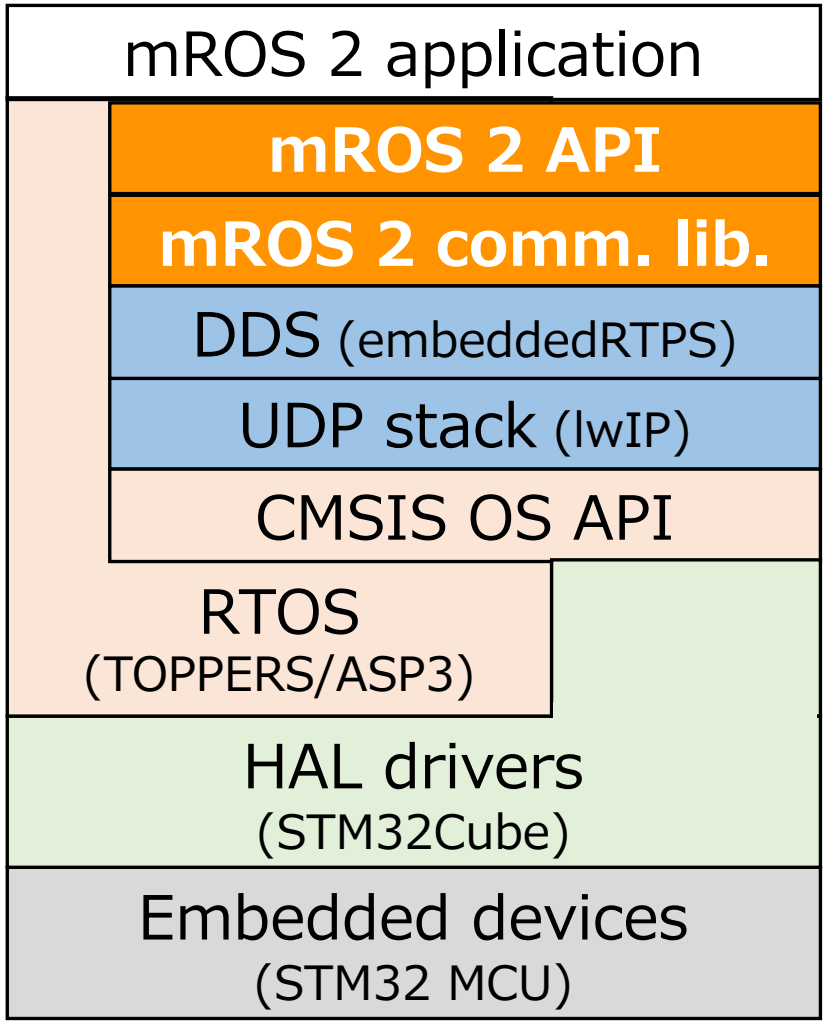
- RTPS：DDSの通信プロトコル
  - SPDP/SEDP：通信相手の探索と通信経路の確立を自律的に行う
- micro-ROS：ROS 2の組込み対応
  - POSIX準拠の複数のRTOSをサポート
  - Micro-XRCE-DDS を利用
  - ➢RTPS通信の仲介に **agent** が必要

詳細は rosjp #43 アーカイブにて,,,



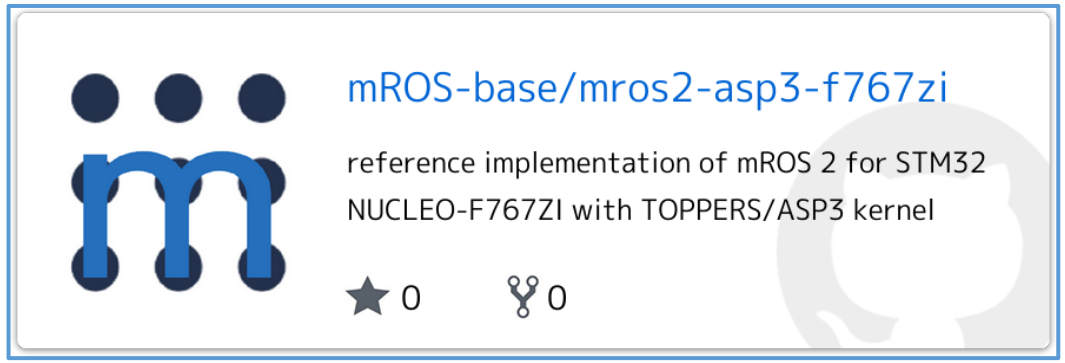
agent無用のROS 2軽量実行環境

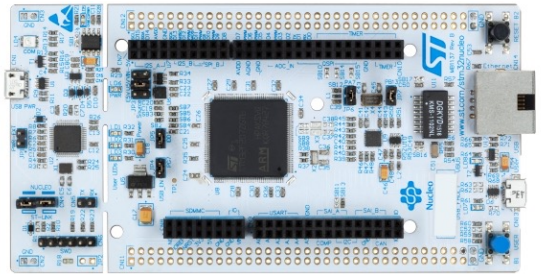
- mROS 2 API & comm. lib.
  - rclcpp にサブ準拠のAPIを提供
  - pub/sub通信機能を提供
- DDS/UDP: embeddedRTPS
  - C++による組込み向けのRTPS軽量実装
  - lwIP, Micro-CDR, FreeRTOS(?) を利用
  - FastDDS 2.3.1との疎通確認済み
- RTOS: (ひとまず)TOPPERS/ASP3
  - 必要なCMSIS OS APIのラッパ層を整備



Getting started!!

```
$ git clone --recursive ¥
  https://github.com/mROS-base/mros2-asp3-f767zi
$ cd mros2-asp3-f767zi/workspace
$ make app=echoreply_string
```



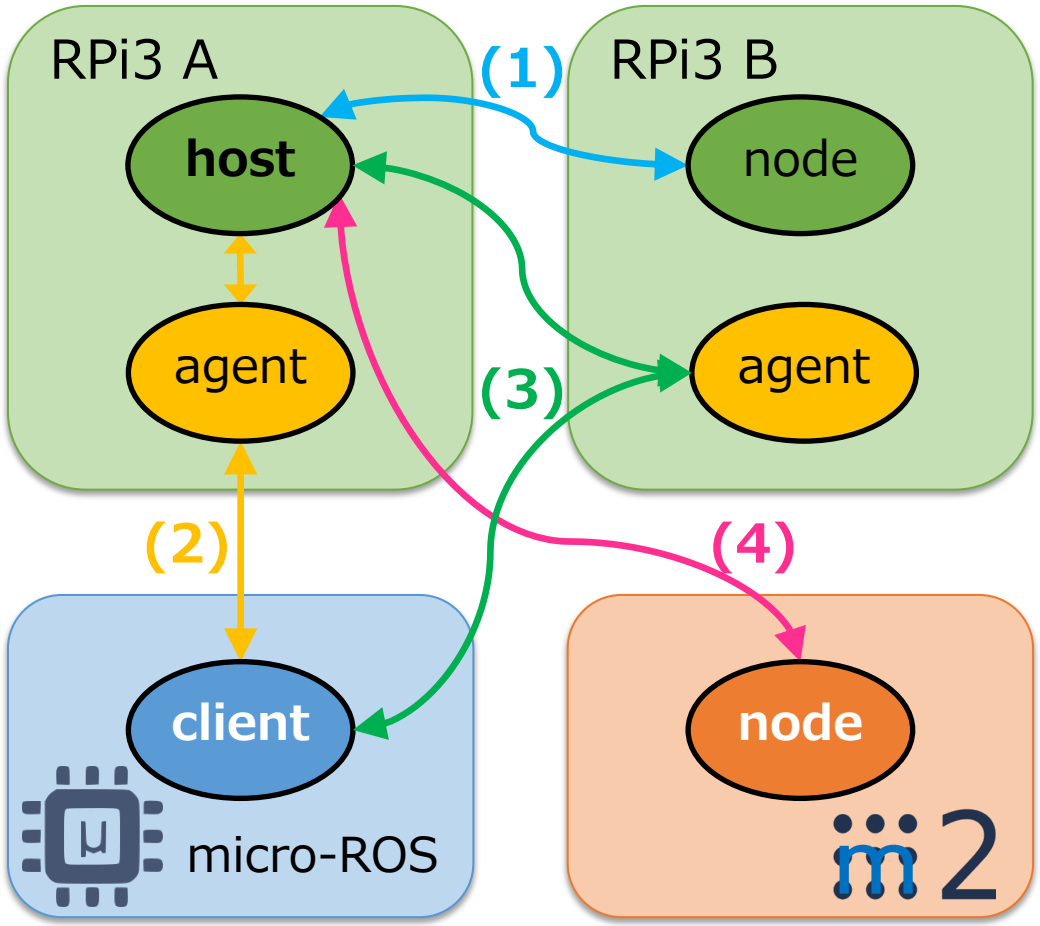


mROS-base/mros2

```
#include "app.h"
#include "mros2.h"
#include "std_msgs/msg/string.hpp"

#include "stm32f7xx_nucleo_144.h"

mros2::Publisher pub;
mros2::Subscriber sub;

void userCallback(std_msgs::msg::String *msg)
{
  MROS2_INFO("subscribed msg: '%s'", msg->data.c_str());
  MROS2_INFO("publishing msg: '%s'", msg->data.c_str());
  pub.publish(*msg);
}

int main(int argc, char * argv[])
{
  MROS2_INFO("mROS 2 application is started");

  mros2::init(argc, argv);
  MROS2_DEBUG("mROS 2 initialization is completed");
  BSP_LED_Toggle(LED1);

  mros2::Node node = mros2::Node::create_node("mros2_node");
  pub = node.create_publisher<std_msgs::msg::String>("to_linux", 10);
  sub = node.create_subscription("to_stm", 10, userCallback);

  MROS2_INFO("ready to pub/sub message");
  mros2::spin();
  BSP_LED_Toggle(LED3);
}

void main_task(void)
{
  main(0, NULL);
}
```

予備的評価 on v0.1-rc

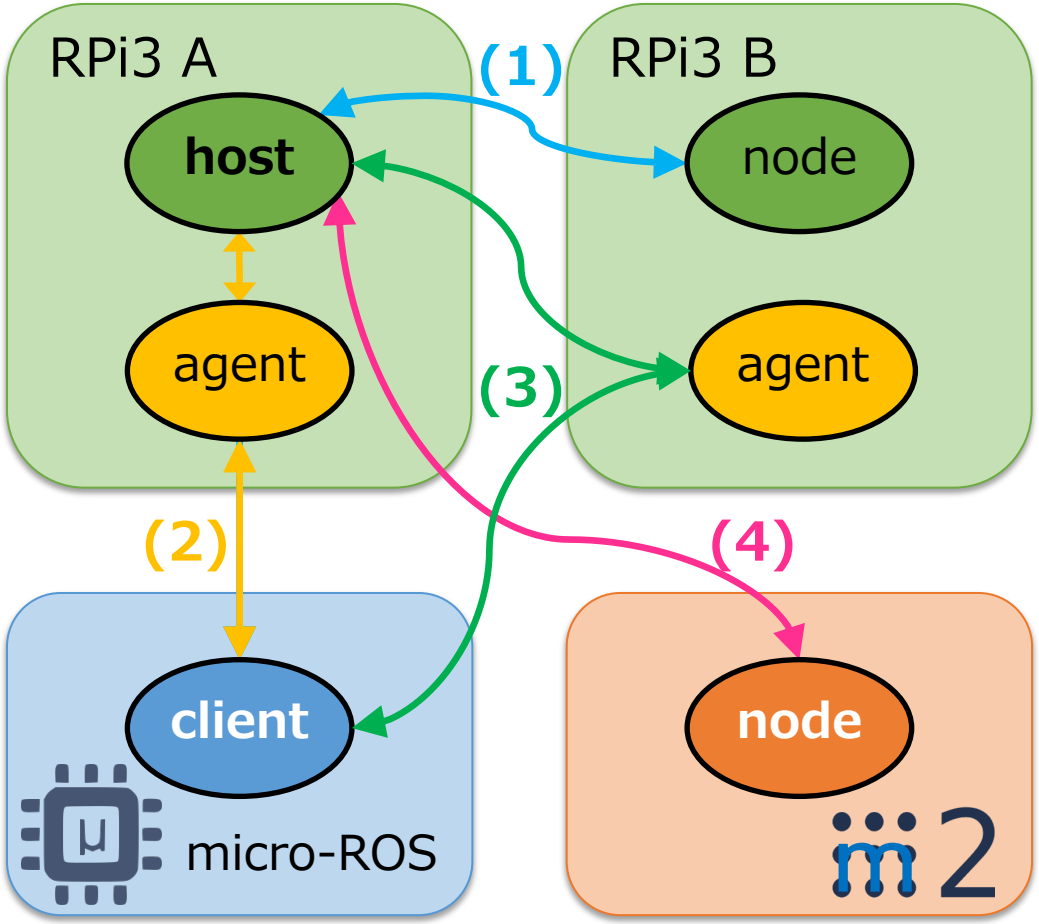
- 評価環境・通信関係



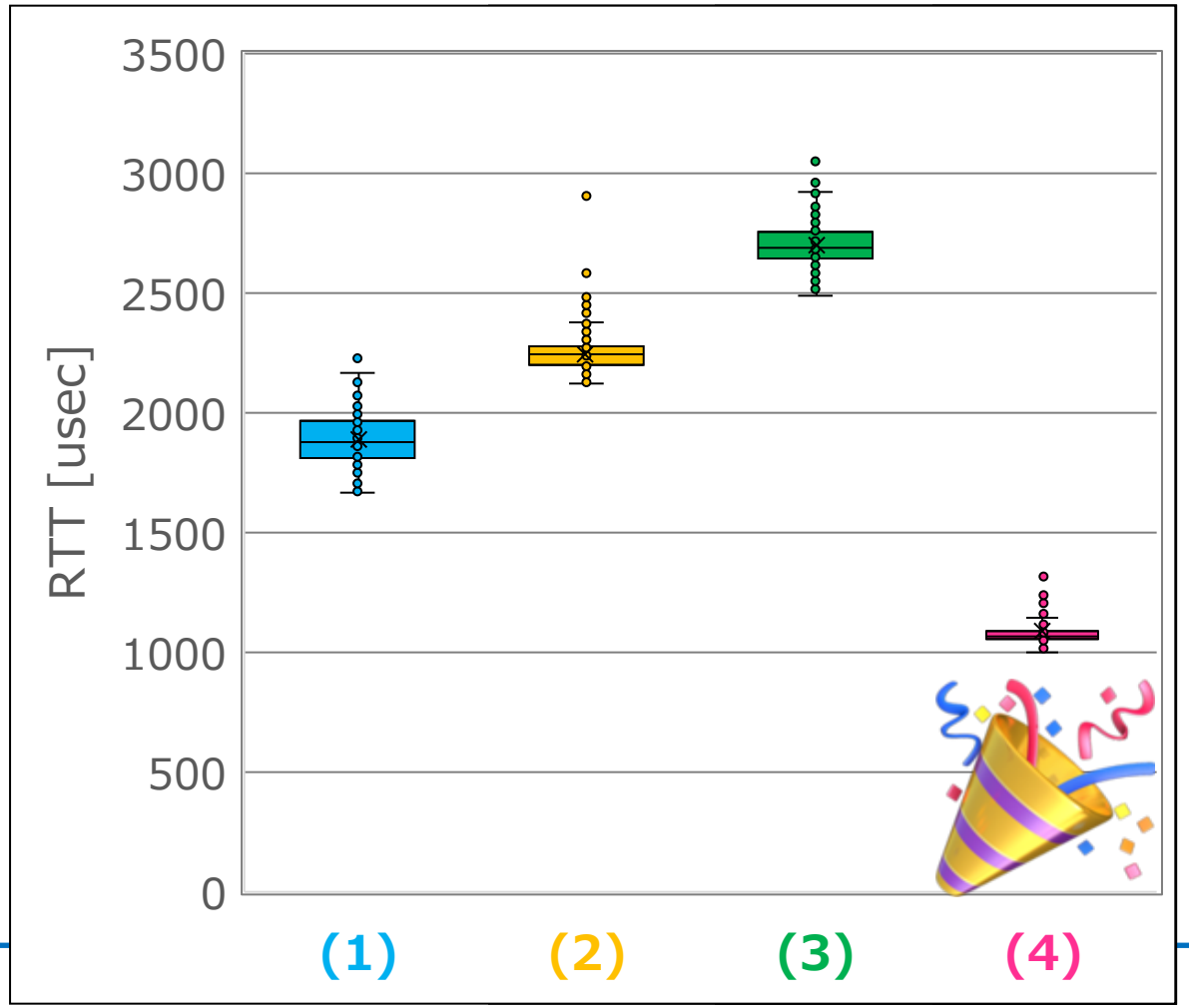
- 各モジュールの性能
  - GPC: Raspberry Pi 3B (1)
    - ✓CA53 1.2GHz x4, 1GB SDRAM
    - ✓Ubuntu 18 & ROS 2 Dashing
  - EMB: STM32 NUCLEO-F767ZI
    - ✓CM7 216MHz, 512KB SRAM
    - a. micro-ROS with NuttX (2) (3)
    - b. mROS 2 with ASP3 (4)
- 評価対象: RTT (round trip time)
  for echo back of Int32 msg
  - 1秒間隔で200回測定

予備的評価 on v0.1-rc

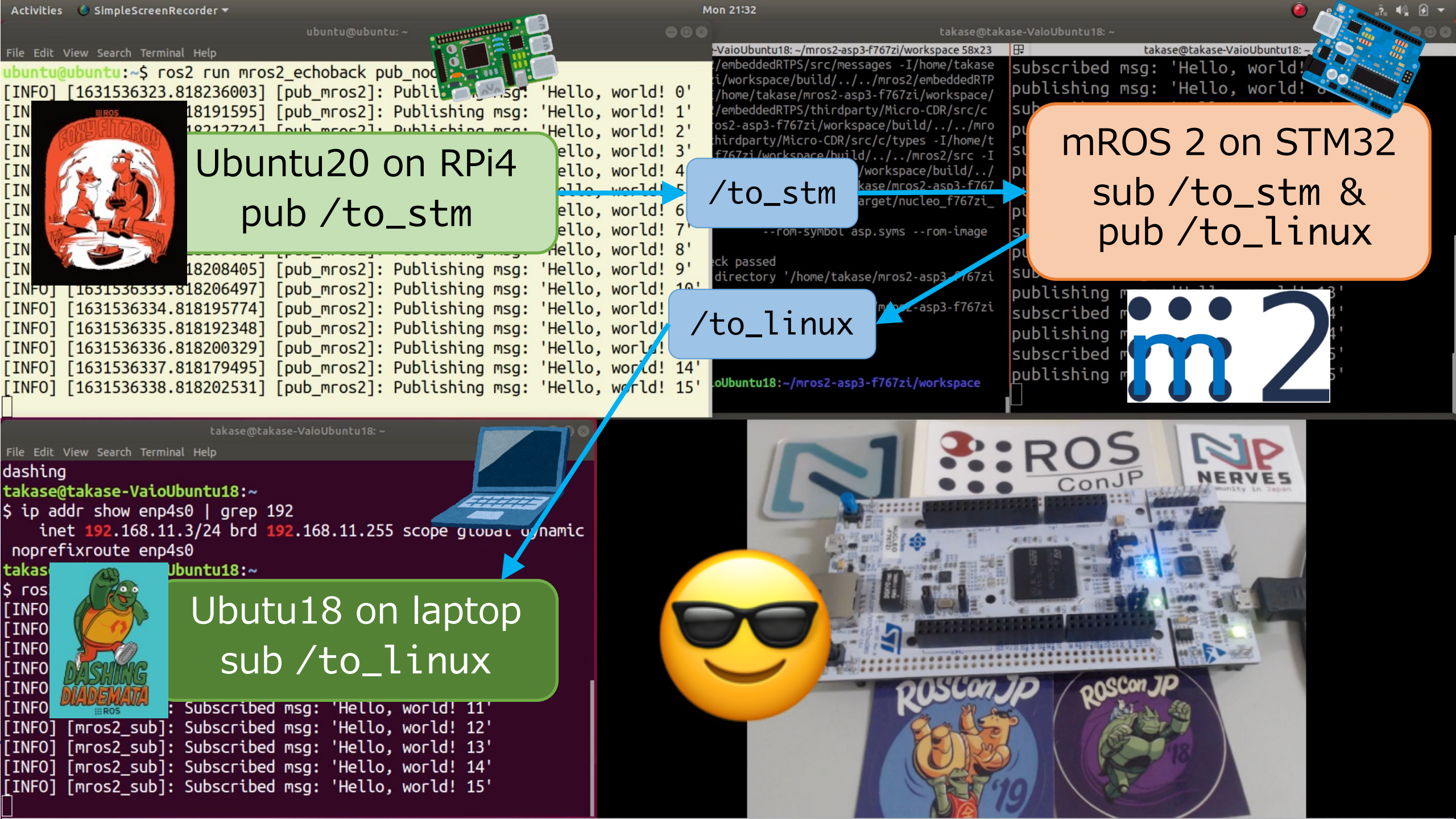
- 評価環境・通信関係



- 評価結果

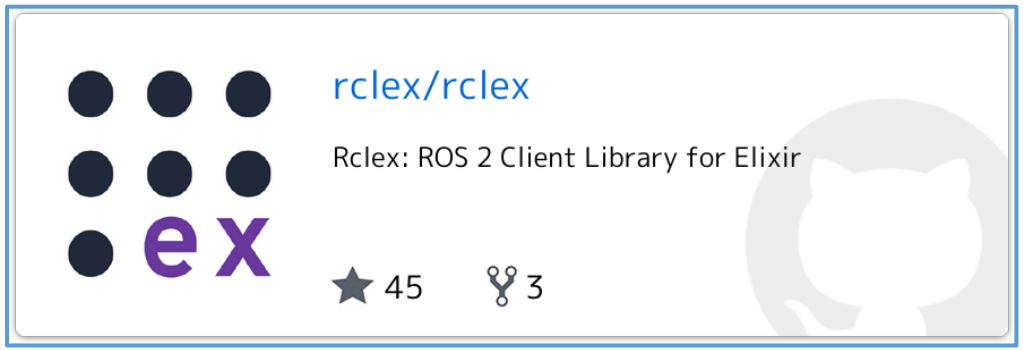


mROS-base/mros2


Activities
SimpleScreenRecorder
Mon 21:32
ubuntu@ubuntu: ~
File Edit View Search Terminal Help
ubuntu@ubuntu:~$ ros2 run mros2_echoback pub_noc
[INFO] [1631536323.818236003] [pub_mros2]: Publi
[INFO] [1631536333.818191595] [pub_mros2]: Publishing msg: 'Hello, world! 1'
[INFO] [1631536333.818208405] [pub_mros2]: Publishing msg: 'Hello, world! 9'
[INFO] [1631536333.818206497] [pub_mros2]: Publishing msg: 'Hello, world! 10'
[INFO] [1631536334.818195774] [pub_mros2]: Publishing msg: 'Hello, world!
[INFO] [1631536335.818192348] [pub_mros2]: Publishing msg: 'Hello, world!
[INFO] [1631536336.818200329] [pub_mros2]: Publishing msg: 'Hello, world!
[INFO] [1631536337.818179495] [pub_mros2]: Publishing msg: 'Hello, world! 14'
[INFO] [1631536338.818202531] [pub_mros2]: Publishing msg: 'Hello, world! 15'
Ubuntu20 on RPi4
pub /to_stm
/to_stm
takase@takase-VaioUbuntu18: ~
takase@takase-VaioUbuntu18: ~/mros2-asp3-f767zi/workspace 58x23
subscribed msg: 'Hello, world!
publishing msg: 'Hello, world!
mROS 2 on STM32
sub /to_stm &
pub /to_linux
/to_linux
takase@takase-VaioUbuntu18: ~
File Edit View Search Terminal Help
dashing
takase@takase-VaioUbuntu18:~
$ ip addr show enp4s0 | grep 192
inet 192.168.11.3/24 brd 192.168.11.255 scope global dynamic
noprefixroute enp4s0
[INFO] [mros2_sub]: Subscribed msg: 'Hello, world! 11'
[INFO] [mros2_sub]: Subscribed msg: 'Hello, world! 12'
[INFO] [mros2_sub]: Subscribed msg: 'Hello, world! 13'
[INFO] [mros2_sub]: Subscribed msg: 'Hello, world! 14'
[INFO] [mros2_sub]: Subscribed msg: 'Hello, world! 15'
Ubutu18 on laptop
sub /to_linux
ROS
ConJP
NERVES
ROSCon JP
ROSCon JP

Epilogue: What's Next??

ROS 2 CL for E^2 すごくない!!?



https://github.com/rclex/rclex

https://github.com/mROS-base/mros2

よろしければ star をお願いします！

まだまだ志は半ば,,,

より"関数型"らしくしたい

Service, Actionに対応したい

ROSIDLをやっていかな,,,

example / case study を増やしたい

Foxyにpubれない,,,

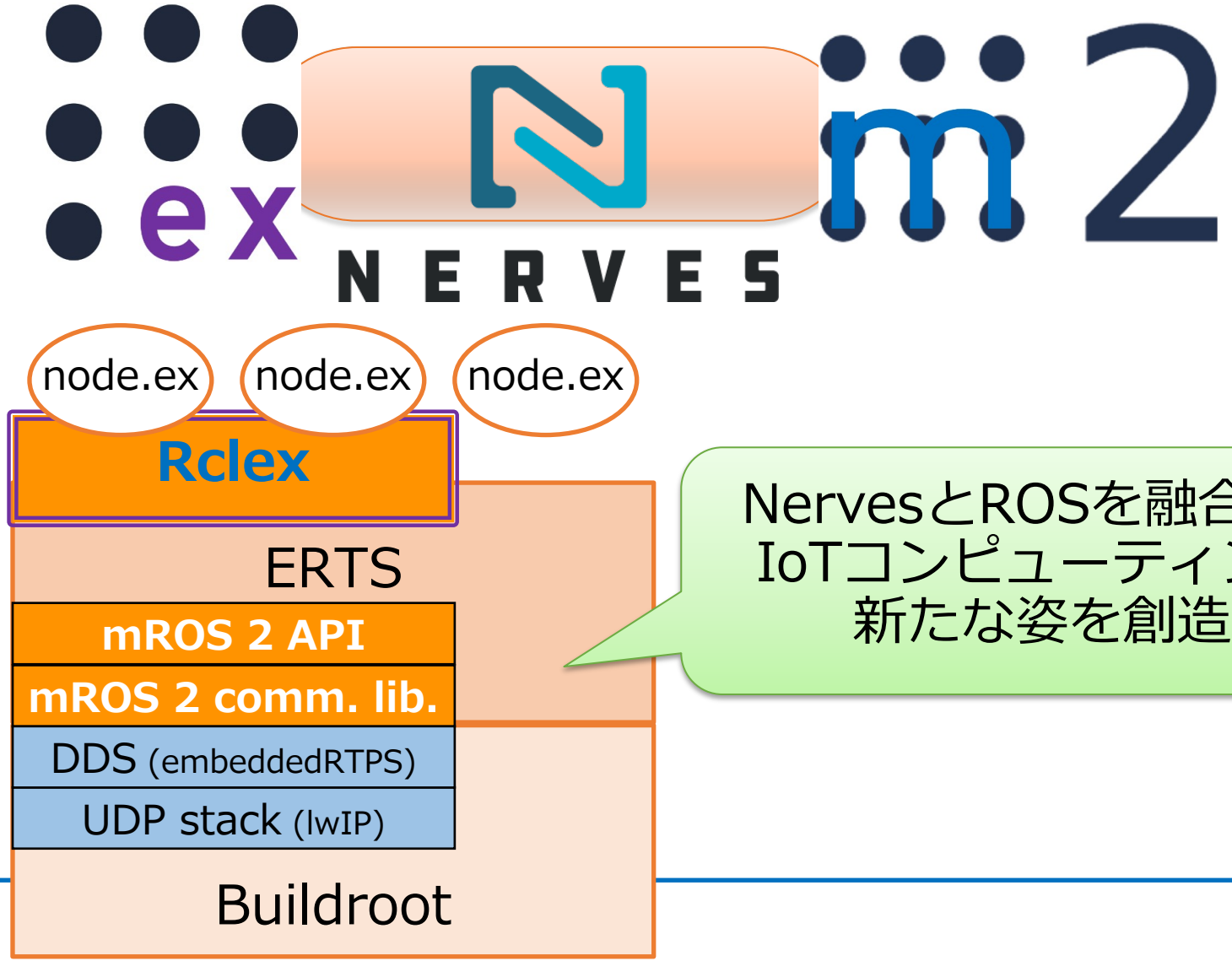
UDP multicastをサポートしたい

対応ボードとカーネルを増やしたい

真面目に性能評価して論文を(ry

Do you ❤ OSS?
Give us your contributions!! 😍

What's Next??


ex
NERVES
m2
node.ex
node.ex
node.ex
Rclex
ERTS
mROS 2 API
mROS 2 comm. lib.
DDS (embeddedRTPS)
UDP stack (lwIP)
Buildroot
NervesとROSを融合して
IoTコンピューティングの
新たな姿を創造！