TOPPERSカンファレンス
2022/06/10

ロボットソフトウェアの組込みデバイス向け軽量実行環境

@takasehideki

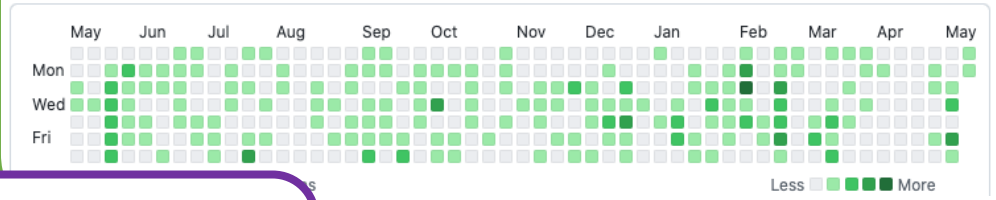
Affiliation

京都大学 KYOTO UNIVERSITY FOUNDED 1897
東京大学 THE UNIVERSITY OF TOKYO
SWEST
ROSCon JP '22
ALGYAN
NERVES JP Nerves Community in Japan

Hobby



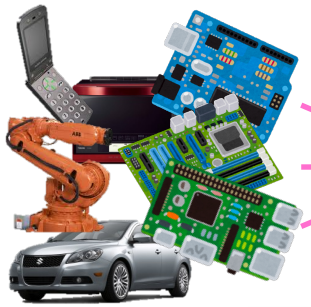
1,650 contributions in the last year



@takasehideki


Mission

組込み／IoTコンピューティング基盤を支える
プラットフォーム技術と設計方法論

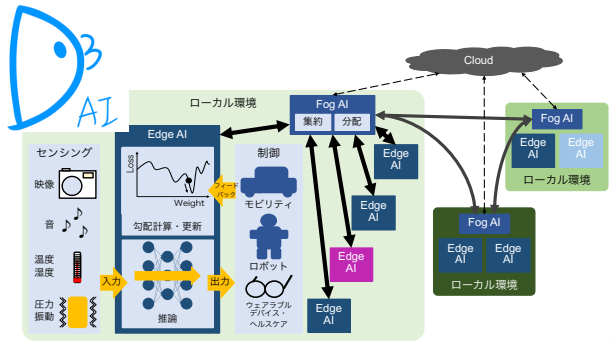


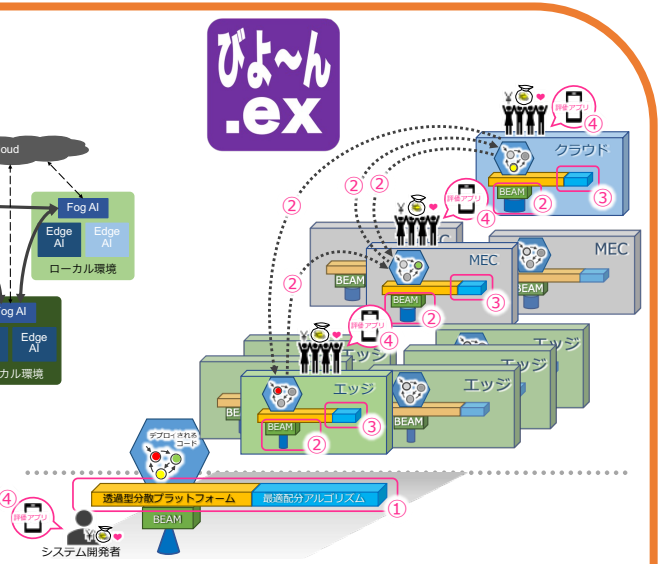
あらゆる
モノ・コトと
ヒトを繋げる
設計基盤の
確立

最適化　**設計技術**

良いモノ を **誰でも楽に**
つくることができる世界へ

Recent R&D

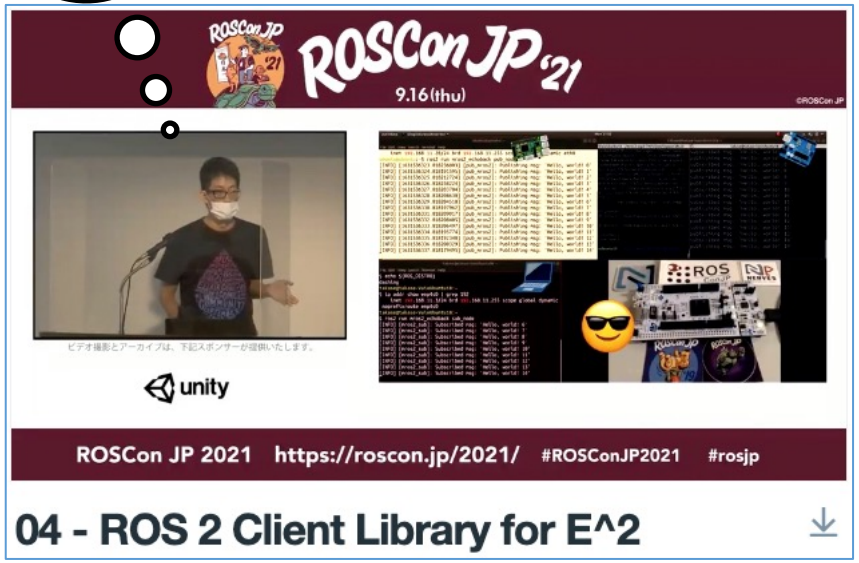




mROS 2??

https://www.stmcu.jp/design/thirdparty/os/91883/

このとき
v0.1.4,,,



https://vimeo.com/638040779/8a10335711

Agenda

- **はじめに：背景と目的**
- **mROS 2**の紹介
  - –方針と設計要件の整理
  - –ソフトウェア構成
  - –対応するプラットフォーム
  - –mROS 2ノードの実装例
- 評価・議論：通信性能・バイナリサイズ・Pros & Cons
- まとめと今後の計画
- 補足：DDS/RTPS・タスク構成と通信処理の動作フロー

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

ROS (Robot Operating System)

ロボットソフトウェアの開発を加速するプラットフォーム

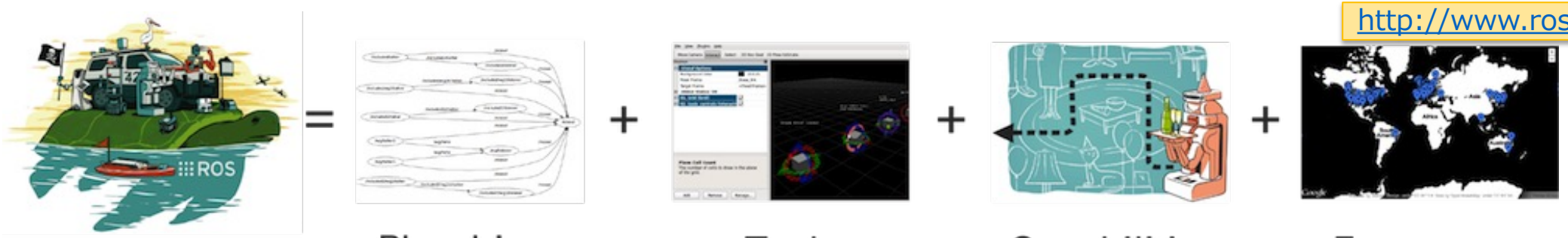
http://www.ros.org/about-ros/

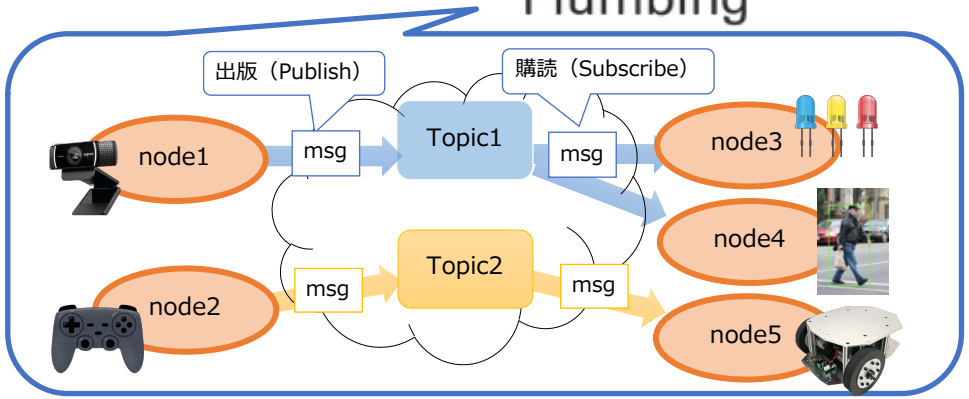
- **ROS (ROS 1)**：2007年11月に最初の正式リリース
  - OSS資産や情報，対応ロボットが豊富にある
  - 最終版 Noetic の EOL は2025年3月まで
- **ROS 2**：絶賛成長中の次世代版
  - ROSのコンセプトを引き継いで再設計されたバージョン
  - 通信層にOMG標準の**DDS (Data Distribution Service)** を採用

ROS (Robot Operating System)

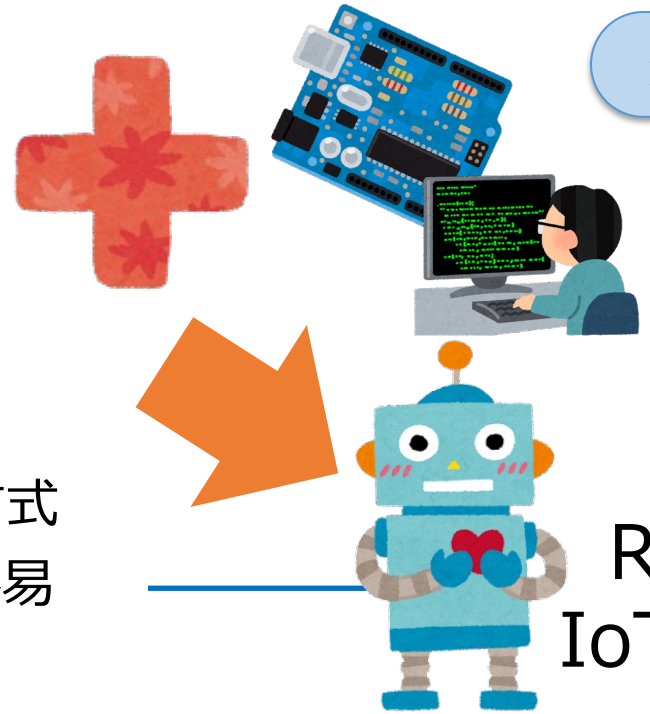
ロボットソフトウェアの開発を加速するプラットフォーム

http://www.ros.org/about-ros/





- ROSの本質は**通信**にあり
  - ROSノード (機能単位) の疎な結合方式
  - 登録・変更・削除・配置・復旧が容易
  - 基本は**Topic**を介した出版購読型

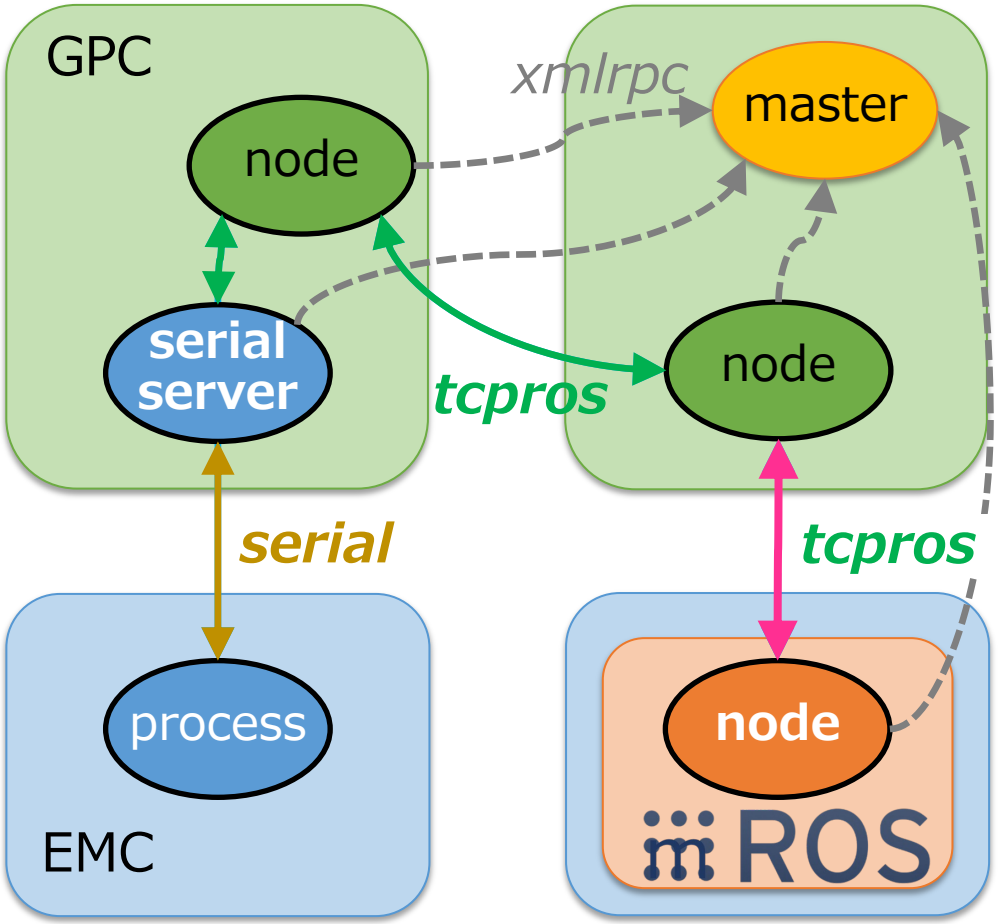


組込み技術の導入

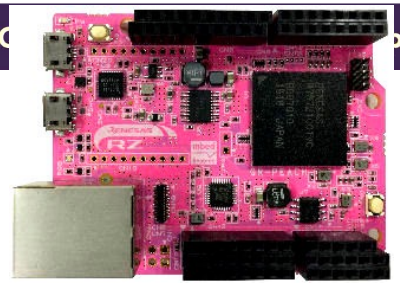
- **通信性能**の向上
  - 応答性：遅延時間
  - リアルタイム性：遅延変動
- **消費電力**の削減

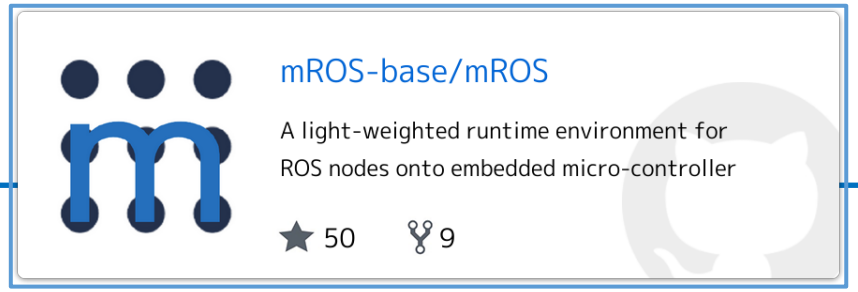
ROS2/DDS通信技術の
IoT分野への展開にも期待

昔話：ROS 1の通信方式と組込み対応

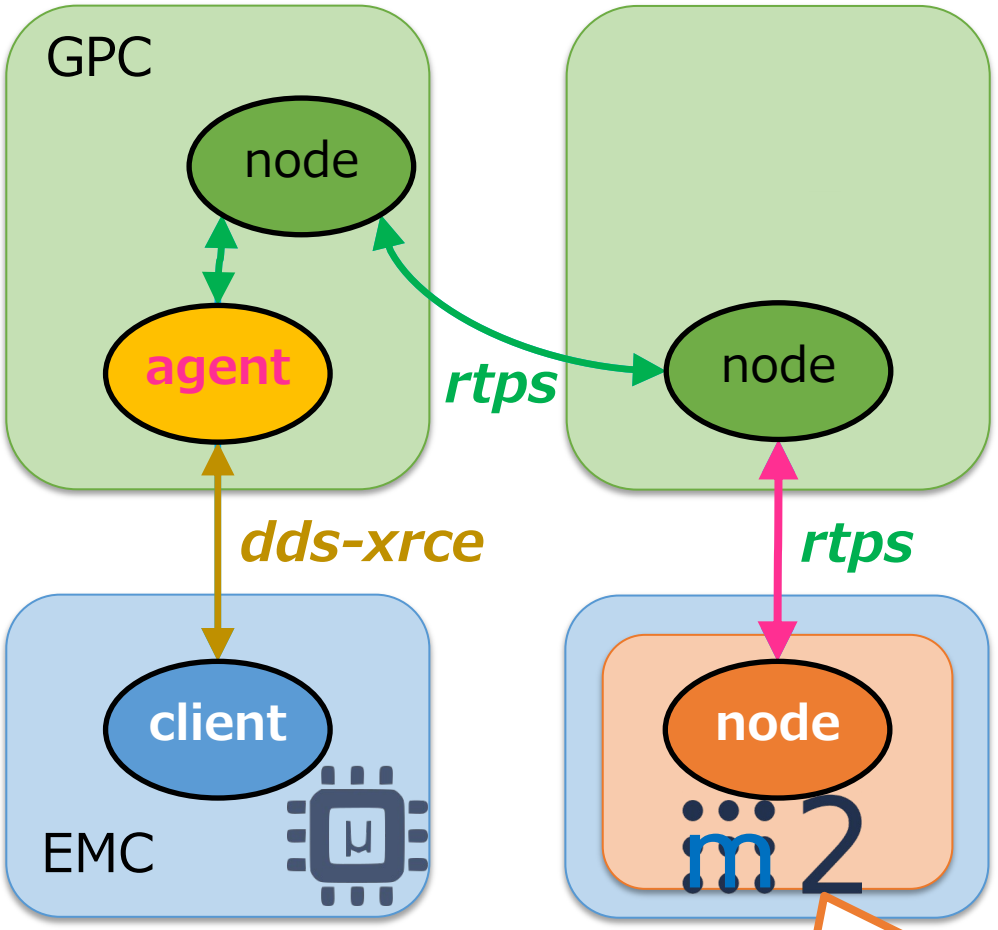


※GPC: General Purpose Computer
EMC: Embedded Micro-Computer





そして時代は⁝2に!!



※GPC: General Purpose C
EMC: Embedded Micro-C

agent無用で
ROS 2通信！

- RTPS：DDSの通信プロトコル
  - SPDP/SEDP：自律的に通信相手と通信経路を探索／確立（master不要）
  - (ROS 2自体が)汎用OS上での稼働を想定

→ 組込み技術の導入

  - エッジ処理の応答性向上や消費電力削減
- 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

→ 単一障害点の発生と通信性能の劣化

本研究開発の目的と貢献

組込みデバイス向けの高効率なROS 2通信方式と
メモリ軽量な実行環境の確立

– 中規模の組込みデバイス上で実行されるプログラムが，
汎用デバイス上のROS 2ノードと自律的に通信できるようにする
– 通信性能に優れた分散ロボットシステムの構築に貢献する

- 提案：

- 本研究における通信性能の定義
  - 応答性：通信処理の遅延時間
  - リアルタイム性：遅延時間の変動
  - ✓いずれも小さいほど良い

Agenda

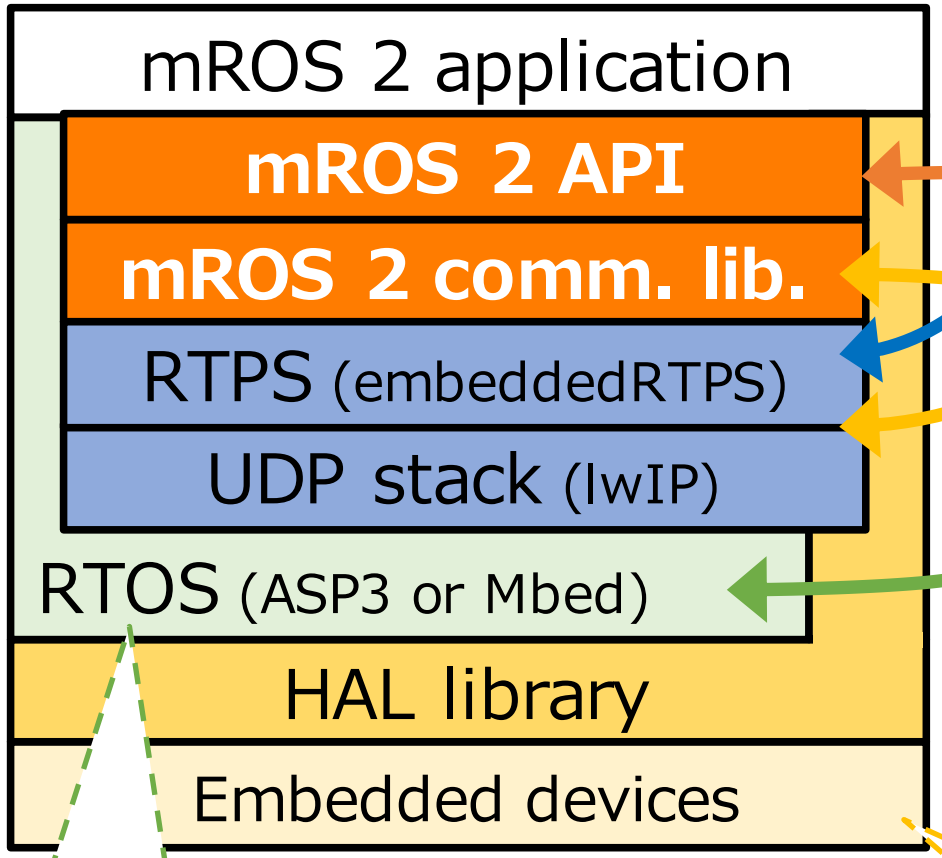
- はじめに：背景と目的
- **mROS 2の紹介**
  - 方針と設計要件の整理
  - ソフトウェア構成
  - 対応するプラットフォーム
  - mROS 2ノードの実装例
- 評価・議論：通信性能・バイナリサイズ・Pros & Cons
- まとめと今後の計画
- 補足：DDS/RTPS・タスク構成と通信処理の動作フロー

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

方針と設計要件の整理

通信の自律性の確保 仲介の仕組みは不要とする	⇒	RTPSの仕様に則った通信の実現 通信性能の低減を下げる
効率的な通信処理の実現 中規模な組込みデバイスでの動作	⇒	C/C++での組込み向けの実装 プロセッサ数百MHz / メモリ数MB程度
メモリ軽量性の実現 リアルタイム性の向上も考慮	⇒	高品質なリアルタイムOSの採用 通信性能の向上にも寄与
基本的な通信方式の実現 開発生産性の観点も考慮	⇒	基本型に対応した出版購読APIの提供 rclcppとの互換性を考慮

ソフトウェア構成



RTPSの仕様に則った通信の実現
通信性能の低減を下げる

C/C++での組込み向けの実装
プロセッサ数百MHz / メモリ数MB程度

高品質なリアルタイムOSの採用
通信性能の向上にも寄与

基本型に対応した出版購読APIの提供
rclcppとの互換性を考慮

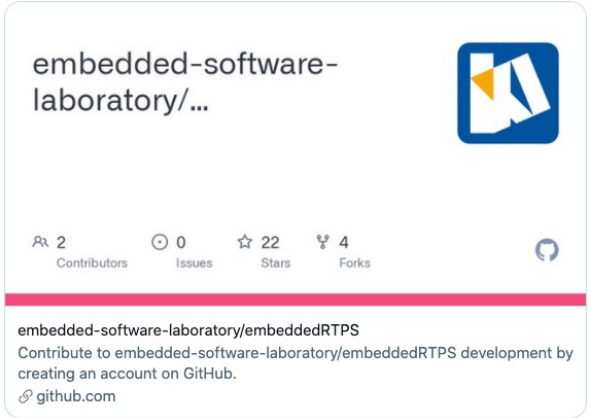
ASP3ではCMSIS
互換レイヤを整備

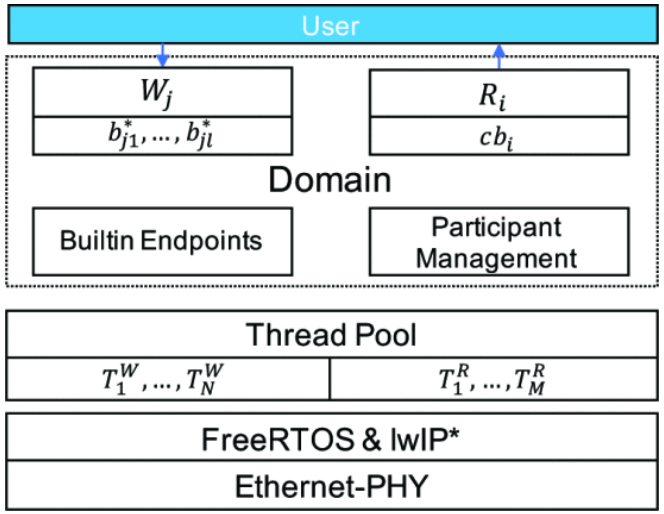
多種デバイスへの
展開の容易化も考慮

mROS-base/mros2

補足：embeddedRTPS [A. Kampmann+ ITSC'2019]

- C++実装による組込み向けのRTPSスタック
  - UDP/IP には lwIP (Raw Mode) を採用
  - シリアライズに eProsima Micro-CDR を利用
- 主な機能と利点
  - Discovery: SPDP と SEDP の機能を提供
  - Interoperability: FastDDS 2.3.1 との疎通確認済み
  - QoS Policies: reliable と best-effort に対応
  - UDP Multicast: PR#4 より対応 (STM32版には未反映)
  - Message size: lwIP バッファサイズまで対応可能

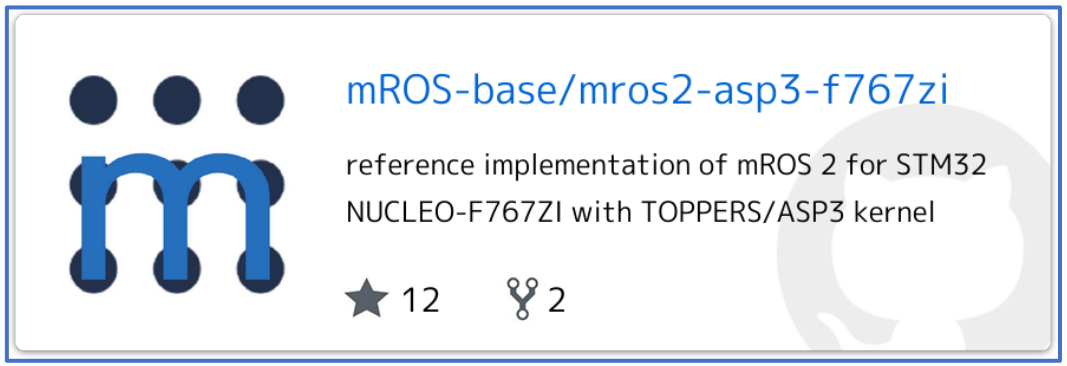


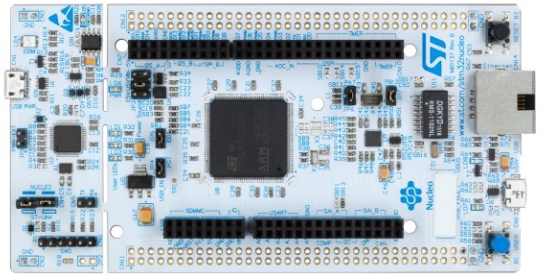


Platform Variation

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

```cpp
#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);
}
```

論よりRUN!!

https://twitter.com/takasehideki/status/1505066116921524228

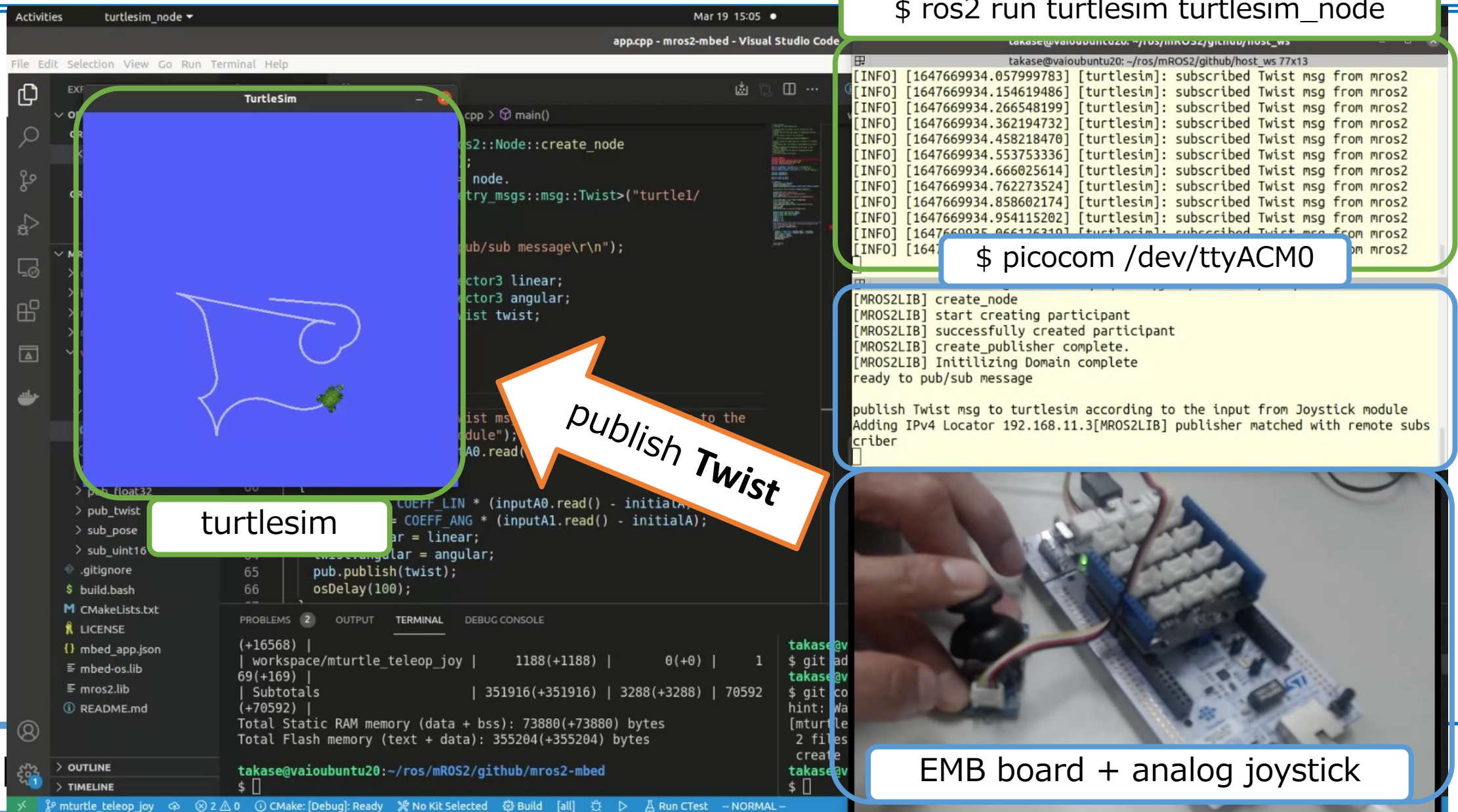
$ ros2 run turtlesim turtlesim_node

```
takase@vaioubuntu20: ~/ros/mROS2/github/host_ws 77x13
[INFO] [1647669934.057999783] [turtlesim]: subscribed Twist msg from mros2
[INFO] [1647669934.154619486] [turtlesim]: subscribed Twist msg from mros2
[INFO] [1647669934.266548199] [turtlesim]: subscribed Twist msg from mros2
[INFO] [1647669934.362194732] [turtlesim]: subscribed Twist msg from mros2
[INFO] [1647669934.458218470] [turtlesim]: subscribed Twist msg from mros2
[INFO] [1647669934.553753336] [turtlesim]: subscribed Twist msg from mros2
[INFO] [1647669934.666025614] [turtlesim]: subscribed Twist msg from mros2
[INFO] [1647669934.762273524] [turtlesim]: subscribed Twist msg from mros2
[INFO] [1647669934.858602174] [turtlesim]: subscribed Twist msg from mros2
[INFO] [1647669934.954115202] [turtlesim]: subscribed Twist msg from mros2
```

$ picocom /dev/ttyACM0

```
[MROS2LIB] create_node
[MROS2LIB] start creating participant
[MROS2LIB] successfully created participant
[MROS2LIB] create_publisher complete.
[MROS2LIB] Initilizing Domain complete
ready to pub/sub message

publish Twist msg to turtlesim according to the input from Joystick module
Adding IPv4 Locator 192.168.11.3[MROS2LIB] publisher matched with remote subs
criber
```



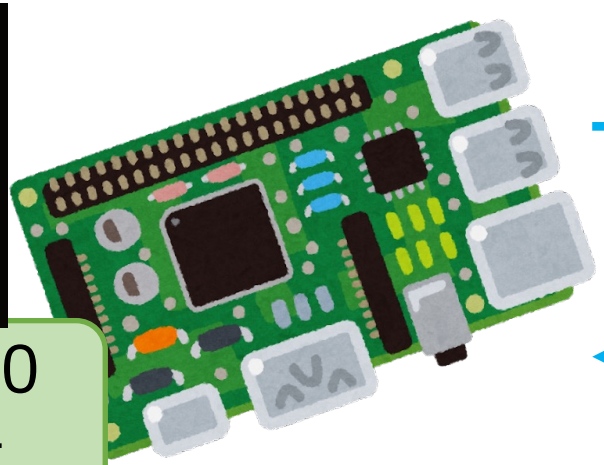
Agenda

- はじめに：背景と目的
- **mROS 2**の紹介
  - –方針と設計要件の整理
  - –ソフトウェア構成
  - –対応するプラットフォーム
  - –mROS 2ノードの実装例
- **評価・議論**：通信性能・バイナリサイズ・Pros & Cons
- まとめと今後の計画
- 補足：DDS/RTPS・タスク構成と通信処理の動作フロー

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

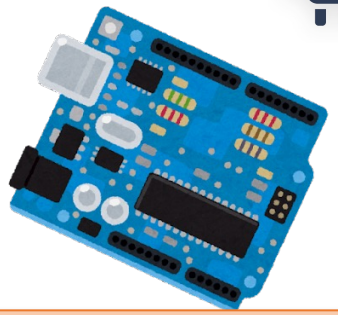
通信性能の評価方法と対象

Ubuntu20
on RPi4



/to_stm

/to_linux



STM32 NUCLEO-F767ZI

UInt16, Twist, String の ping-pong 通信に掛かる時間を測定

rclcpp::WallTimer.get_clock() を使用

	uros-serial	uros-udp	uros-rtps	mros2-asp3	mros2-mbed
API	rclc (galactic)			mROS 2 API v0.3.1	
RTPS	Micro XRCE DDS		embeddedRTPS		
protocol	USART	UDP	RTPS on UDP		
RTOS	FreeRTOS v2			TOPPERS/ASP3	Mbed OS 6
compiler	8.3.1		9.3.1	7.3.1	10.3.1

通信性能の評価結果(1)

✓100回測定
✓単位は [ms]

- UInt16

	uros-serial	uros-udp	uros-rtps	mros2-asp3	mros2-mbed
avg	11.710	2.109	5.182	0.570	0.646
max	17.370	4.240	11.190	0.810	0.940
min	7.590	1.900	1.940	0.490	0.560
std.p	3.094	0.244	2.684	0.067	0.081

- Twist

	uros-serial	uros-udp	uros-rtps	mros2-asp3	mros2-mbed
avg	19.530	2.304	5.508	0.593	0.703
max	25.510	11.250	9.860	0.850	0.880
min	15.590	2.050	2.610	0.520	0.640
std.p	3.666	0.904	1.551	0.065	0.042

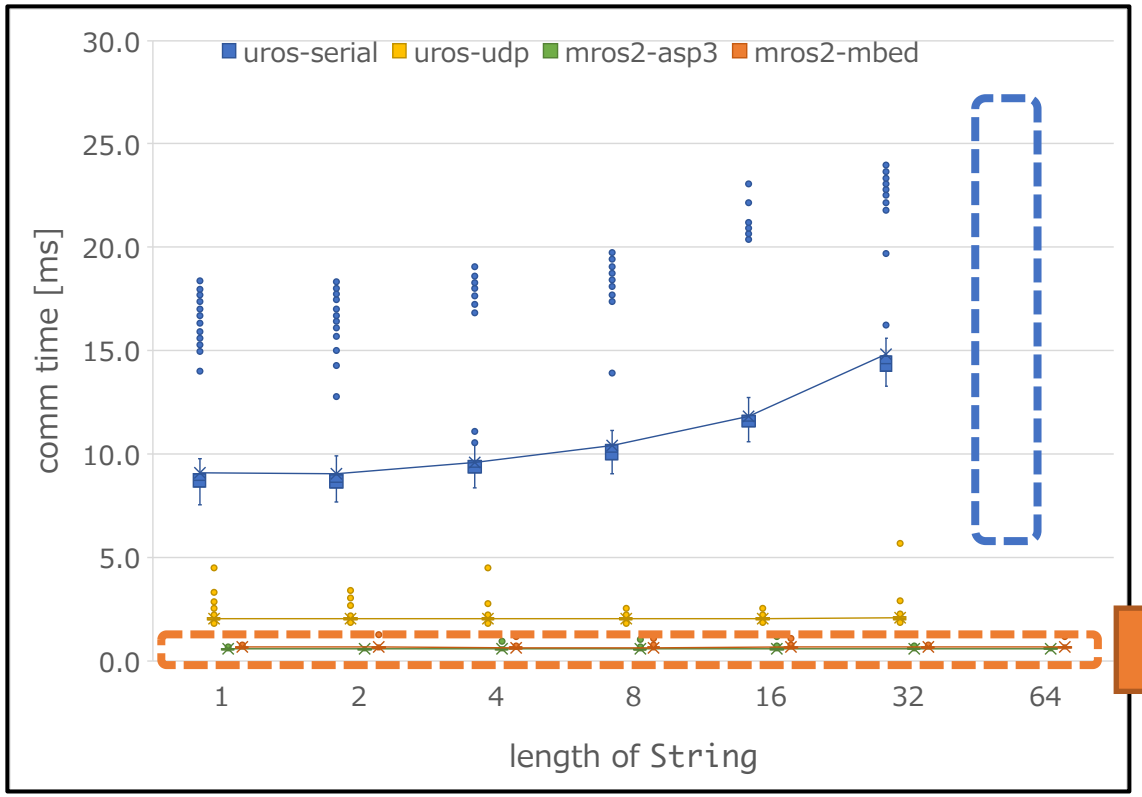
✓mros2- のほうが明らかに優れている
  ➢最大値(max) = 応答性
  ➢標準偏差(std.p) = リアルタイム性

✓uros-rtps では出版の連続回数に応じて
通信時間が増加していく傾向がある

通信性能の評価結果(2)

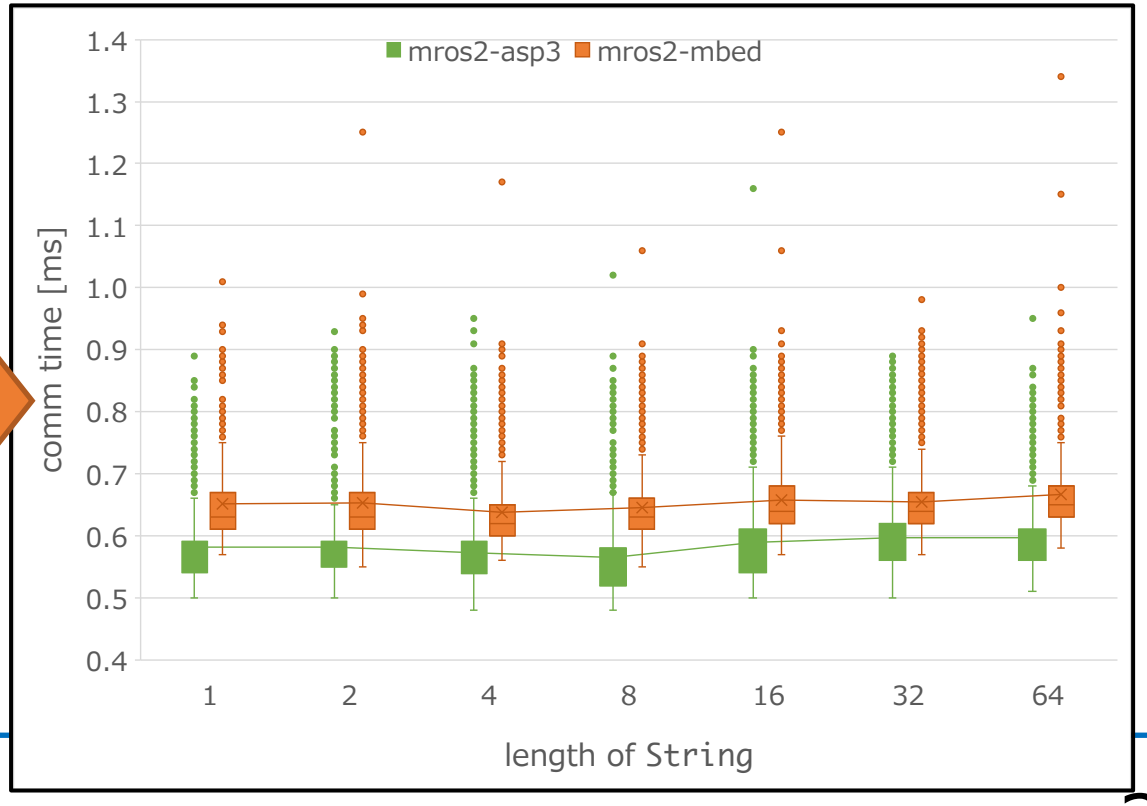
✓文字列長ごとに
1,000回ずつ

- String



➢ uros-rtps は安定的に通信できず

➢ -serial, -udp は64字では通信できず



✓mros2- 同士では
asp3 がより安定的

メモリサイズの評価結果

✓Twist のバイナリで測定
✓単位は [Byte]

- memory size

	uros-serial	uros-udp	uros-rtps	mros2-asp3	mros2-mbed
text	209,836	233,656	174,752	90,551	393,312
data	356	356	576	16,632	3,336
bss	110,280	108,160	282,016	111,800	70,688
total	320,472	342,172	457,334	225,983	469,336

✓mros2-asp が最も優位
  ➢data の増加は静的OSの方針による
  ➢bss はUDPバッファサイズ等の調整で最適化
✓機能が限定的であることも考慮する必要がある
  ➢uros- はQoS制御やservice通信などROS 2互換機能を備える

議論：Pros and Cons

	uros-serial	uros-udp	uros-rtps	mros2-asp3	mros2-mbed	summary (weakness of mros2)
latency	×	△	×	◎	○	mros2-同士ではasp3がやや良い
real-time	×	△	×	◎	○	-mbedはまれに外れ値が観測されることがある
stability	△	△	×	△	○	uXRCEはagentが必須　-rtpsは接続難あり -asp3は初回通信時に欠損することあり
mem. size	△	△	×	○	×	特に-asp3は静的OSの方針に完全に基づく
variety	◎	○	○	×	△	uros-は多様なチップ・ボードに対応 -asp3は1種のみ　-mbedは選択肢多め
productivity	△	△	△	○	◎	mros2-ではC++でノード実装可能 コンパイル環境の準備も容易
perfection	◎	◎	○	△	△	uros-はROS 2機能に完全準拠 mros2-はトピック通信のみ
community	◎	◎	△	×	×	uros-は世界規模！　mros2-はこれから,,,

Agenda

- はじめに：背景と目的
- **mROS 2**の紹介
  - –方針と設計要件の整理
  - –ソフトウェア構成
  - –対応するプラットフォーム
  - –mROS 2ノードの実装例
- 評価・議論：通信性能・バイナリサイズ・Pros & Cons
- **まとめと今後の計画**
- 補足：DDS/RTPS・タスク構成と通信処理の動作フロー

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

まとめと今後の計画

mROS 2 組込み向けの軽量実行環境

- –効率的な通信処理のためのソフトウェア構成と動作フローの設計
- –ROS 2への組込み技術の導入の容易化に貢献
- –分散ロボットシステムにおける通信性能と消費電力に寄与

- 今後の計画 (IOT WiP)
  - –embeddedRTPS の multicast 対応版への追従
  - –通信性能と軽量性を維持したままの機能拡張および品質向上
  - –mros2-posix のターゲット追加
    - ✓Ubuntu20 (native & docker) では動作確認済み
    - ✓POSIX コンパチな RTOS に持っていきたい,,,

Check it out!!

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

Please give us the Star!
& your contribution!!

- **support other boards & TOPPERS kernels**
- implement new targets with POSIX compliant RTOS
- support QoS control, Service, Action, ...
- check with Galactic & support Humble??

mROS-bas

9月にハンズオン，開催決定！！

life.augmented

ALGYAN
All Gadget Your Alliance and Network

詳細の続報は
#ALGYAN にて！

TOPPERS/ASP3版で
やれるといい，かも？
しらんけど :D

mROS-base/mros2

圧倒的！感！謝！！

@ken551

@Hibagon1go

@smoriemb

@tmori

本研究の一部は，JST CREST JPMJCR21D2 の支援ならびに
国立研究開発法人情報通信研究機構の委託研究（04001）により
得られたものである.

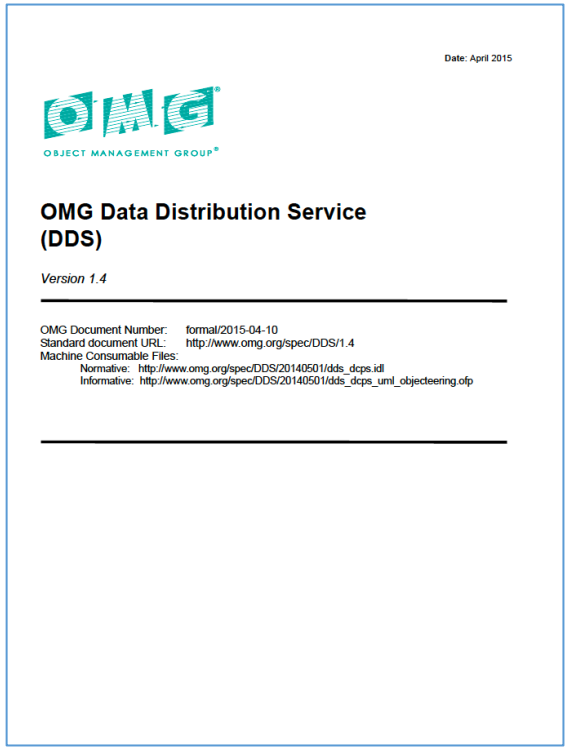
Agenda

- はじめに：背景と目的
- **mROS 2**の紹介
  - –方針と設計要件の整理
  - –ソフトウェア構成
  - –対応するプラットフォーム
  - –mROS 2ノードの実装例
- 評価・議論：通信性能・バイナリサイズ・Pros & Cons
- まとめと今後の計画
- **補足：DDS/RTPS・タスク構成と通信処理の動作フロー**

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

DDS (Data Distribution Service)

- OMG (Object Management Group) で策定された通信仕様
  - peer-to-peer の通信ミドルウェアを実現
  - 最新版は2015年4月公開の Version 1.4
    https://www.omg.org/spec/DDS
- ROS 2での採用による利点
  - ROS 2のメンテコード削減
  - 厳格かつ明確な仕様に依存できる
    - ✓第三者によるレビューや監査が可能
  - 高い互換性のある実装が提供される
    - ✓not “The DDS Wars”, but “DDS Cambrian Explosion” !!


Date: April 2015

OMG

OBJECT MANAGEMENT GROUP®

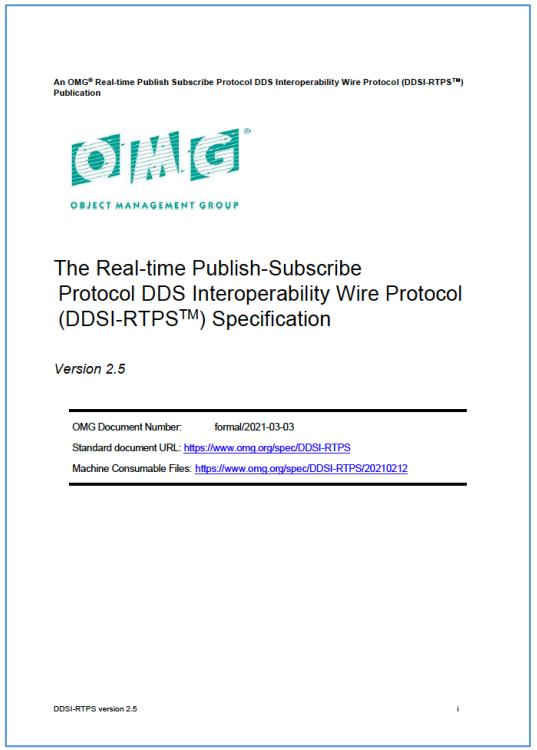
OMG Data Distribution Service (DDS)

Version 1.4

OMG Document Number: formal/2015-04-10
Standard document URL: http://www.omg.org/spec/DDS/1.4
Machine Consumable Files:
Normative: http://www.omg.org/spec/DDS/20140501/dds_dcps.idl
Informative: http://www.omg.org/spec/DDS/20140501/dds_dcps_uml_objecteering.ofp

RTPS (Real-Time Publish-Subscribe)

- DDSの通信プロトコル
  - 正式名称は DDS Interoperability Wire Protocol (DDSI-RTPS)
  - 最新版は2021年3月公開の Version 2.5
    https://www.omg.org/spec/DDSI-RTPS/
- UDP/IP 上に実装される
  - OSI参照モデルのtransport層に位置
  - パケットの到着保証などの不確実性は QoS機能によってカバーする
- 通信相手の探索／通信経路の確立を自律的に行うことができる


An OMG® Real-time Publish Subscribe Protocol DDS Interoperability Wire Protocol (DDSI-RTPS™) Publication

OMG
OBJECT MANAGEMENT GROUP

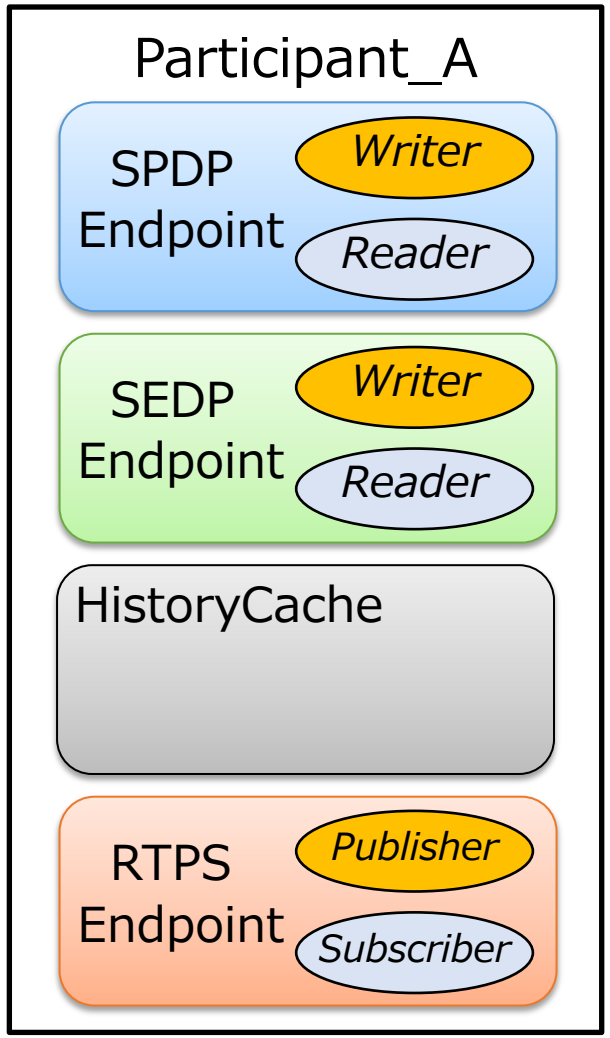
The Real-time Publish-Subscribe Protocol DDS Interoperability Wire Protocol (DDSI-RTPS™) Specification

Version 2.5

OMG Document Number: formal/2021-03-03
Standard document URL: https://www.omg.org/spec/DDSI-RTPS
Machine Consumable Files: https://www.omg.org/spec/DDSI-RTPS/20210212

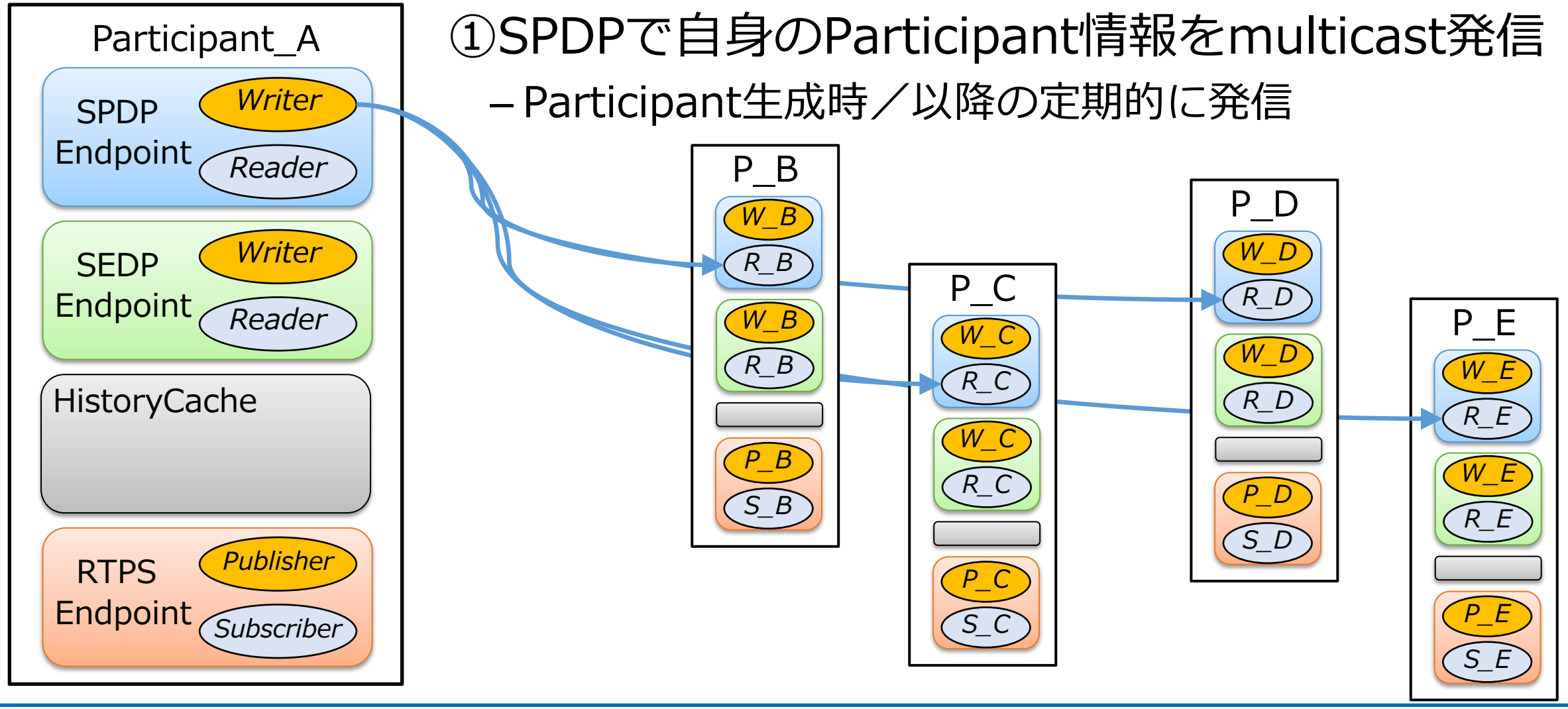
DDSI-RTPS version 2.5 i

RTPS による通信確立のフロー

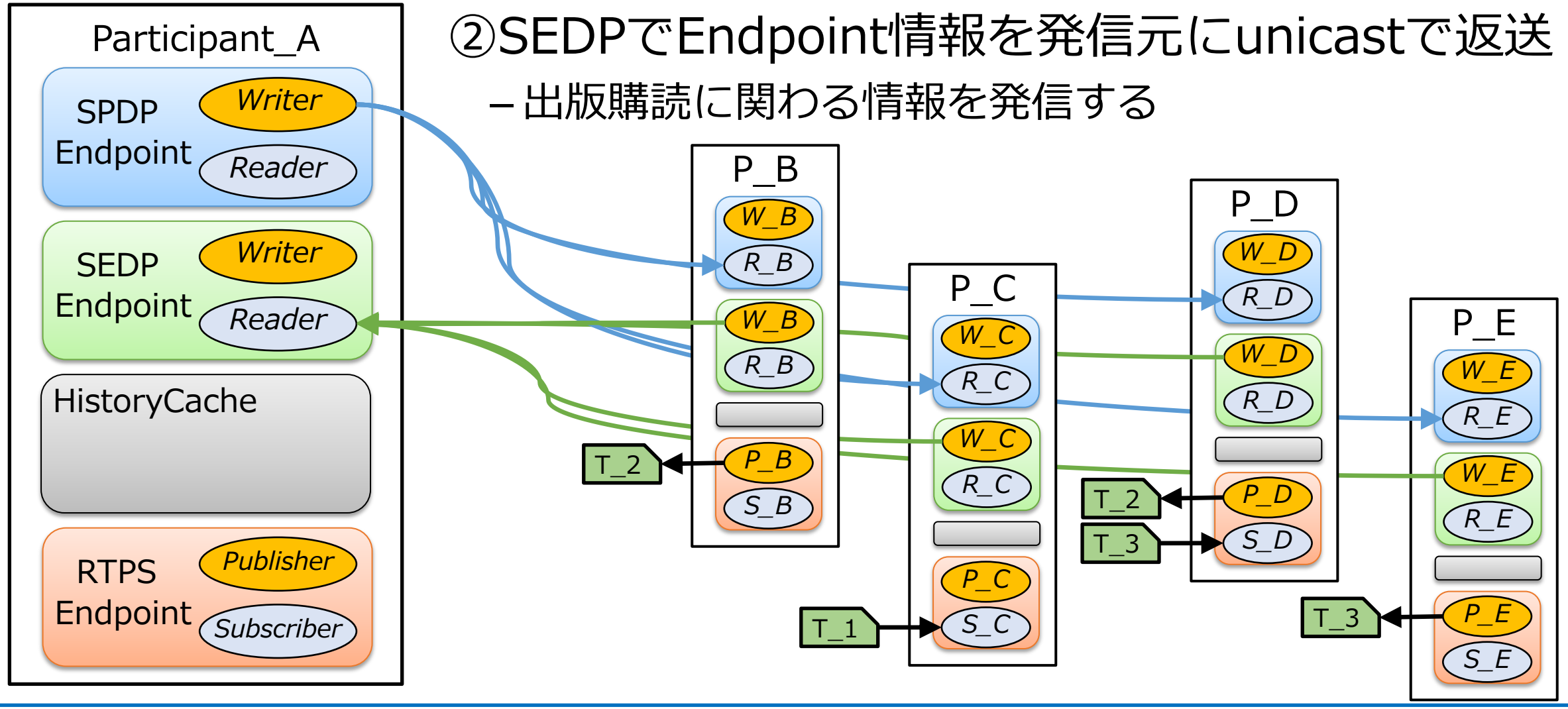


- Participant: ROS 2におけるノード
- SPDP: Simple Participant Discover Protocol
- SEDP: Simple Endpoit Discover Protocol
- Endpoint: 通信端点となるモジュール
  - Writer/Reader: 情報を送信／受信するモジュール
- HistoryCache: 通信履歴を保持するモジュール

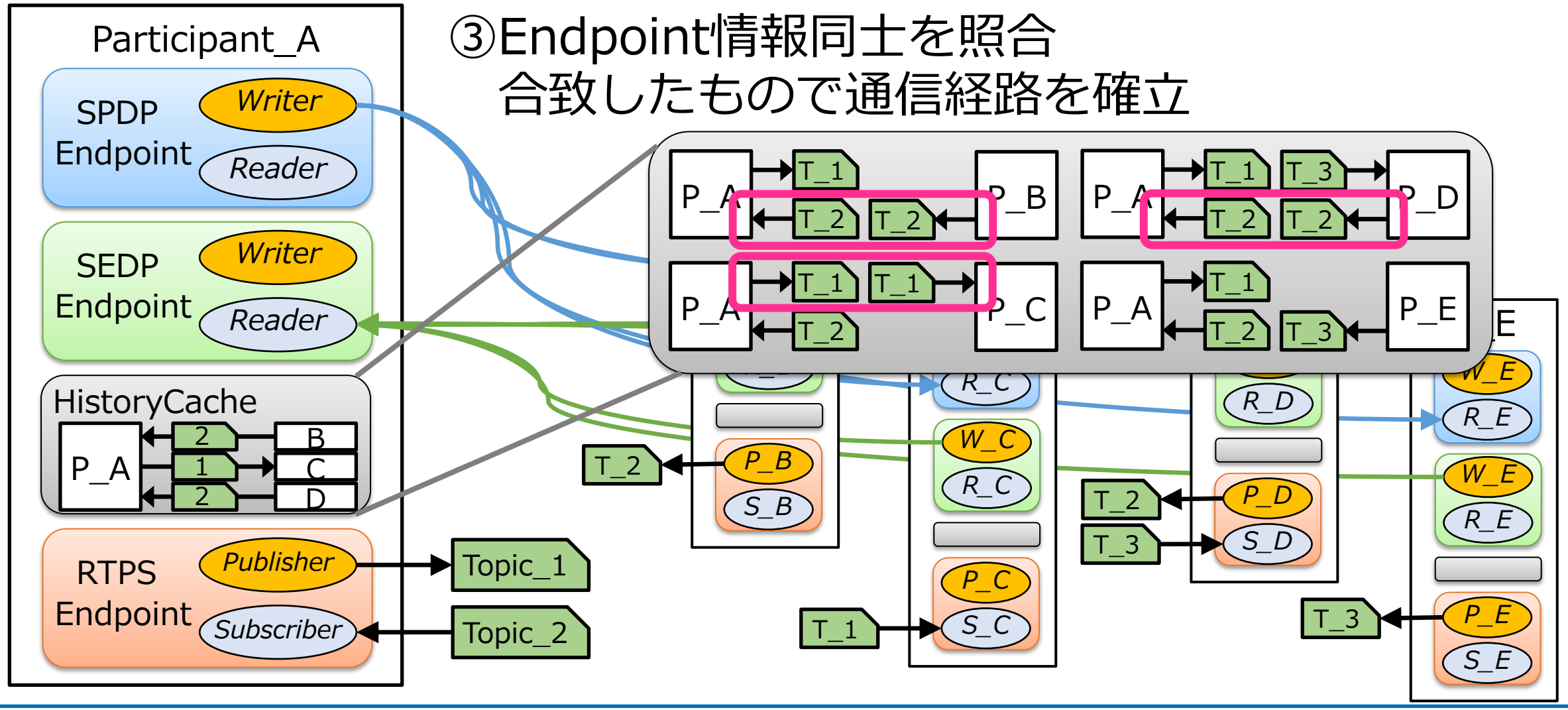
RTPS による通信確立のフロー



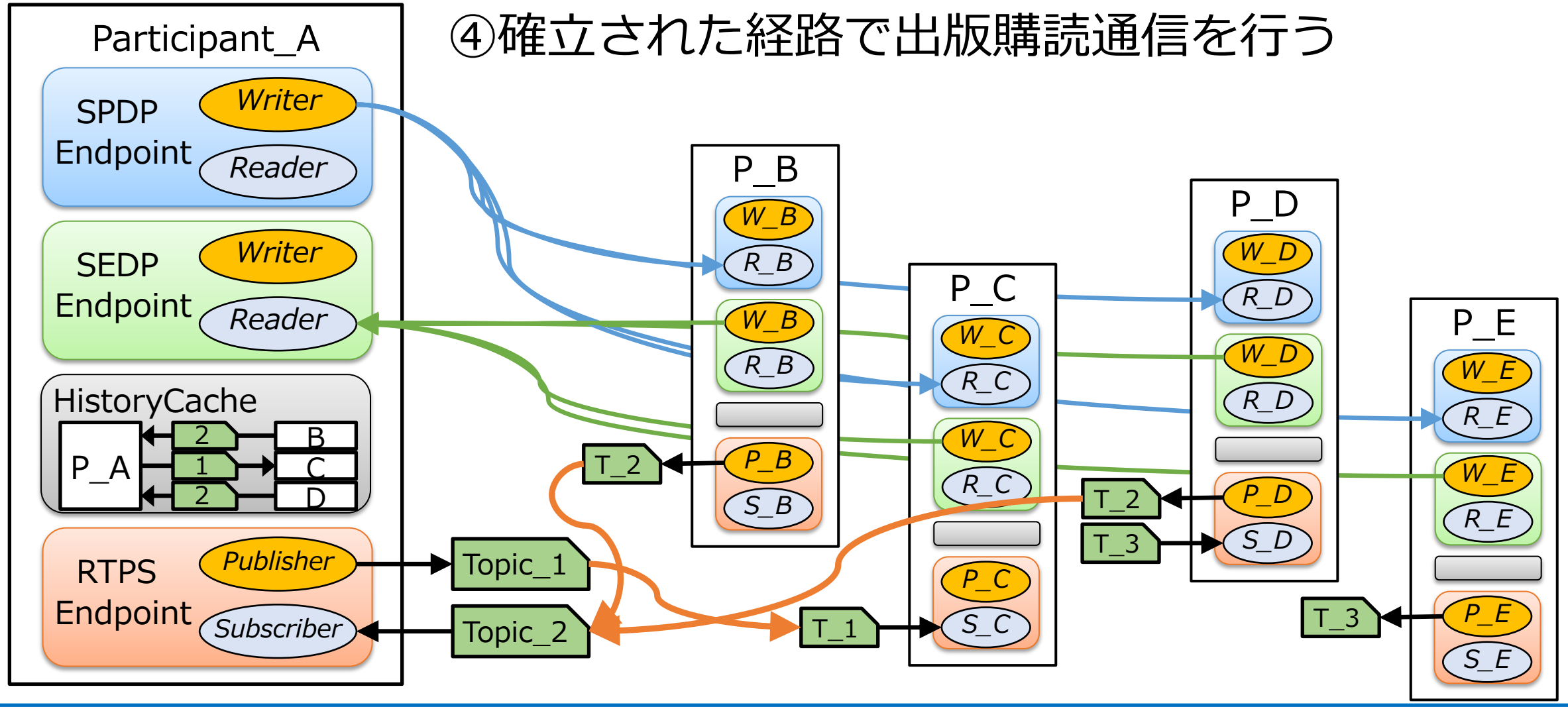
RTPS による通信確立のフロー



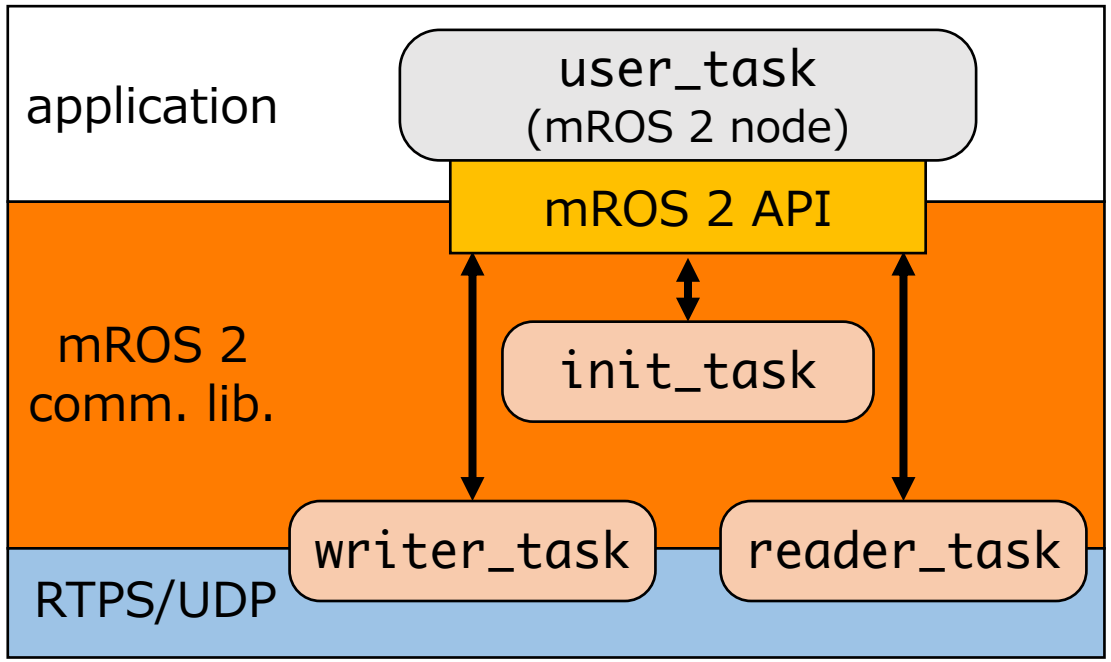
RTPS による通信確立のフロー



RTPS による通信確立のフロー

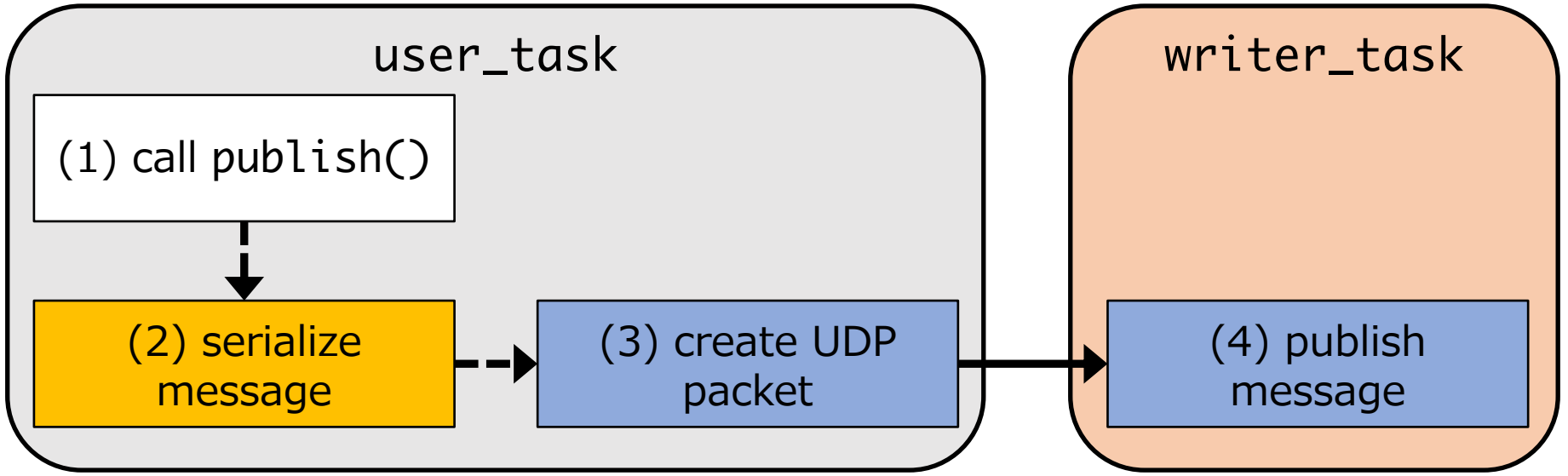


mROS 2 のタスク構成



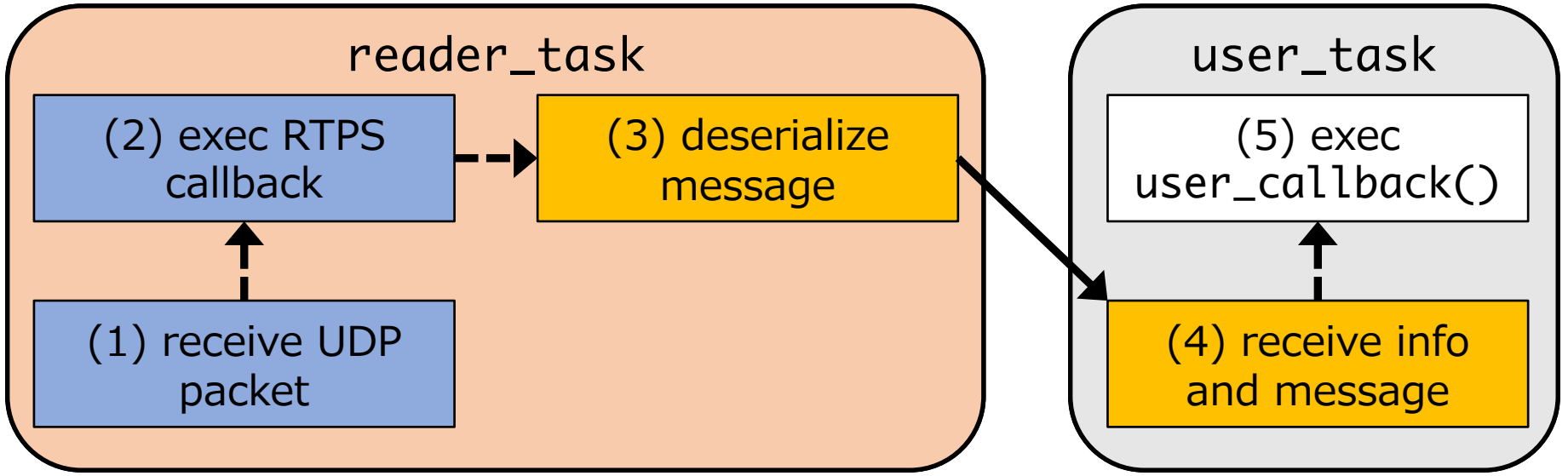
- user_task
  - ROS 2のノードに相当
  - マルチタスクプログラミングも可能
- init_task
  - ノード情報の初期化
  - RTPS Participant として登録
  - RTPS Reader/Writer 機能のためのインスタンスを生成
- write_task / reader_task
  - API に応じて出版購読処理を担う

出版処理の動作フロー



(1) メッセージ情報のオブジェクトのポインタを引数として publish() を呼出し
(2) メッセージをシリアライズしてRTPSに則ったデータ形式に変換
(3) UDPパケットを作成し，writer_task に出版処理を依頼
(4) 出版処理を writer_task 内で実行（user_task と並行処理できる）

購読処理の動作フロー



(1) UDPパケットを reader_task で受信

(2) 初期化時に登録された embeddedRTPS 内のコールバック関数を実行

(3) RTPSパケットをデシリアライズしてメッセージのデータ形式に変換

(4) コールバック関数とメッセージのオブジェクトのポインタを user_task に通信

(5) コールバック関数を user_task のコンテキストで実行