SWEST Summer Workshop on Embedded System Technologies

2024/08/30

SWEST26 セッション~~s5b~~ -> s2c

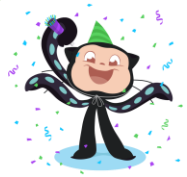
すべてが #Zenoh になる
～柔軟にして軽量～

@takasehideki

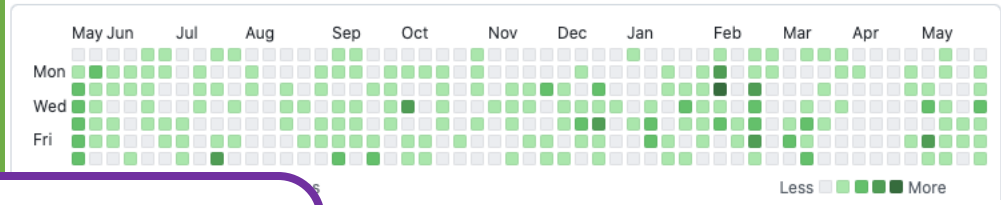
Lab#8, IPC, IST, UTokyo
Computing System Laboratory

Affiliation

Hobby

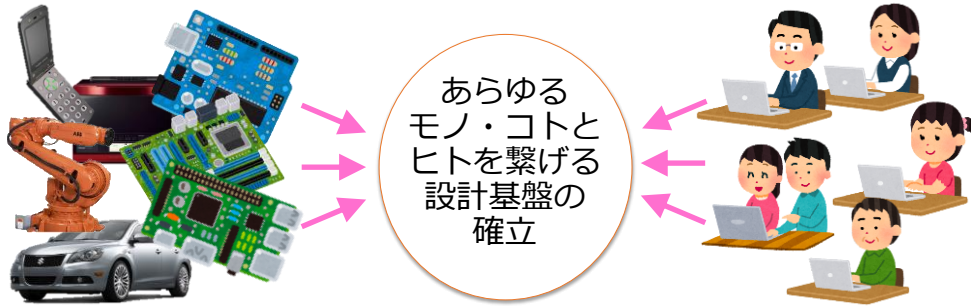


1,728 contributions in the last year



@takasehideki

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

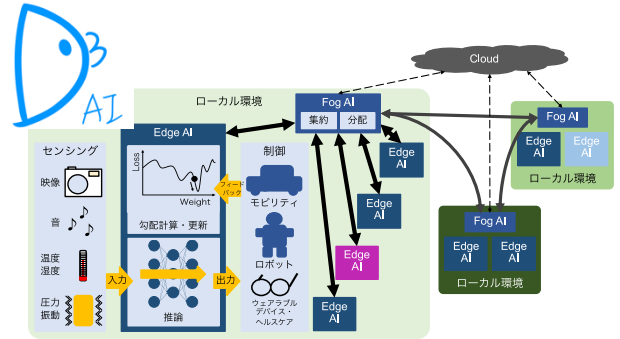


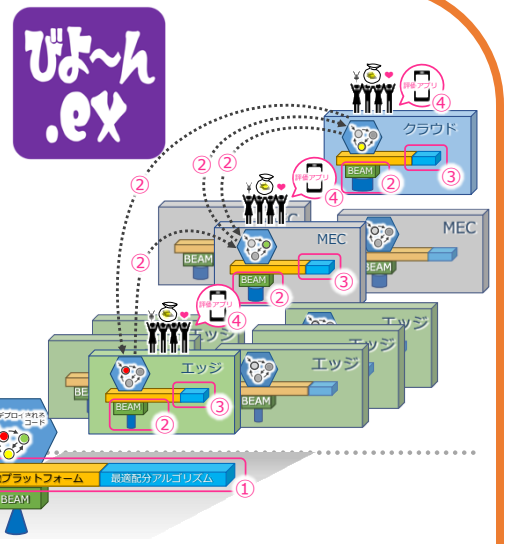
最適化　　**設計技術**

Mission

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





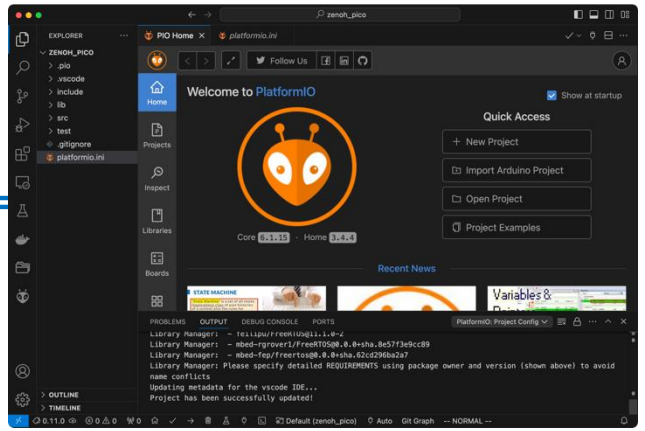
m2ex

箱庭

本セッションの構成

- 前半：チュートリアル (a.k.a Zenoh完全に理解した:D
  - 前口上：IoT Computing と出版購読型通信
  - Zenoh の４つの側面
  - 広がるセカイ：ロボットにもマイコンにも！
- 後半：ハンズオン (a.k.a Zenohまるでわからん;;(
  １：様々な言語でつなげる
  ２：IoT的につなげる

ハンズオンの前準備

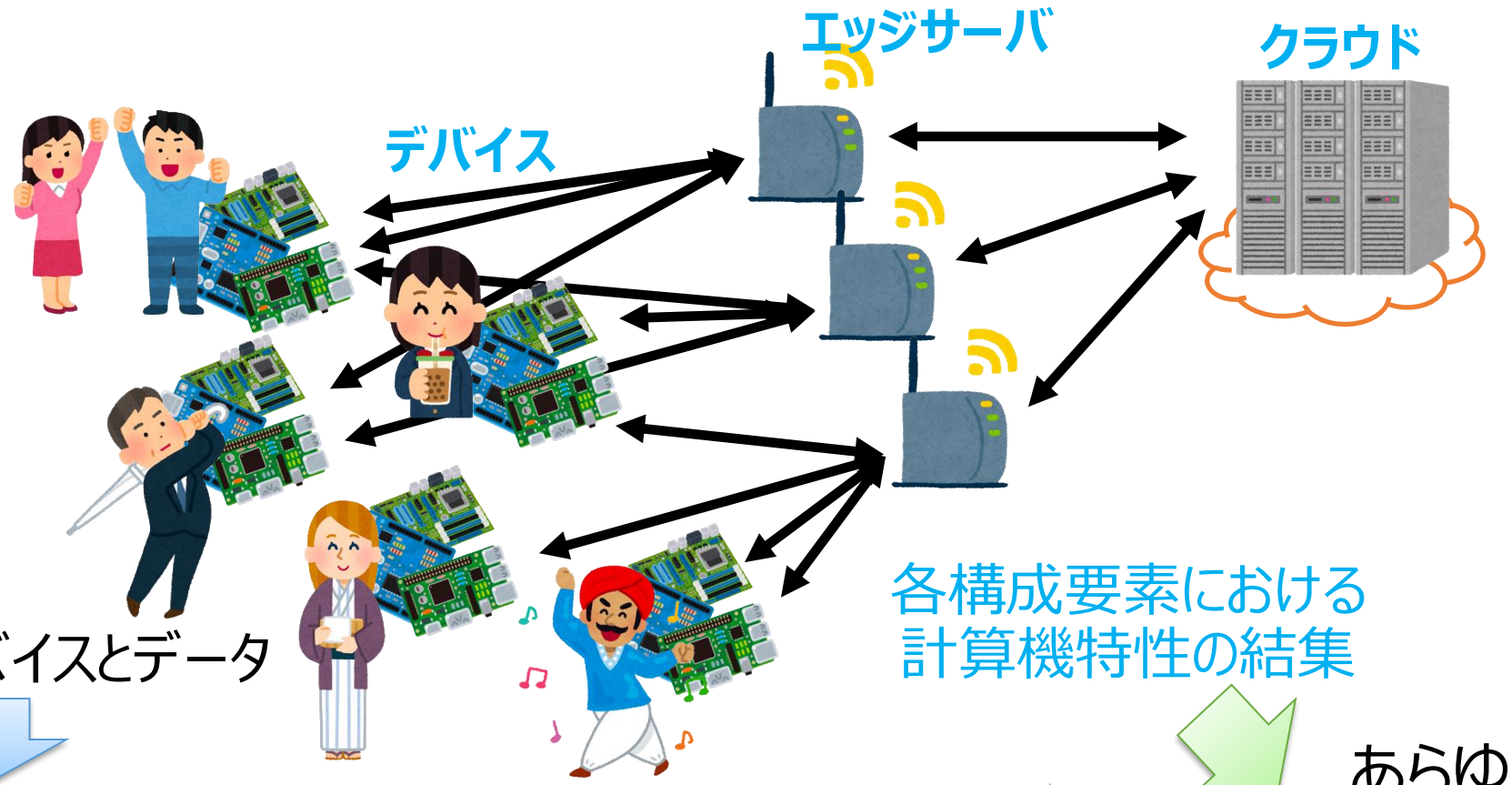


- ハンズオンの前準備
  - 環境準備は済んでいますよね？？
  - 会場参加者：PCを専用のWi-Fi APに接続してください
    - ✓SSID / PASS は会場で案内します
  - ハンズオン用のリポジトリを clone してください

```
git clone --recursive https://github.com/takasehideki/zenoh_swest26_trial
```

    - ✓--recursive 必須！
  - リポジトリにある zenoh_pico/ のディレクトリを VSCode で開く
    - ✓ウインドウ下の “OUTPUT” に Project has been sucessfully updated! が表示されるまで（チュートリアルを聴きながら）待ちます

IoT Computing??



大量のデバイスとデータ

「デバイスレベル並列化」による最適化の時代へ！

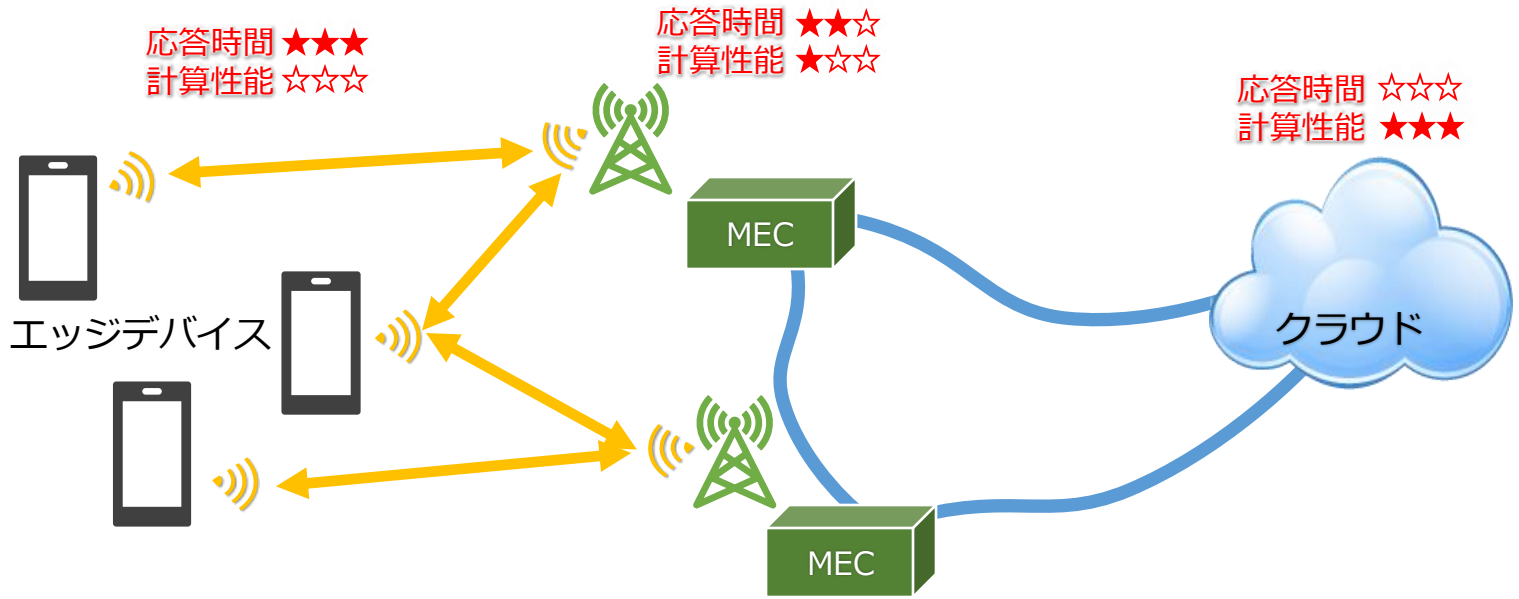
AI/ML技術との密接な統合

各構成要素における計算機特性の結集

あらゆるモノ・コトのネットワーク接続によって
新たな社会的価値を創造

広域分散処理

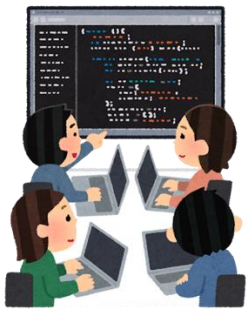
IoT computingの
理想像!!?

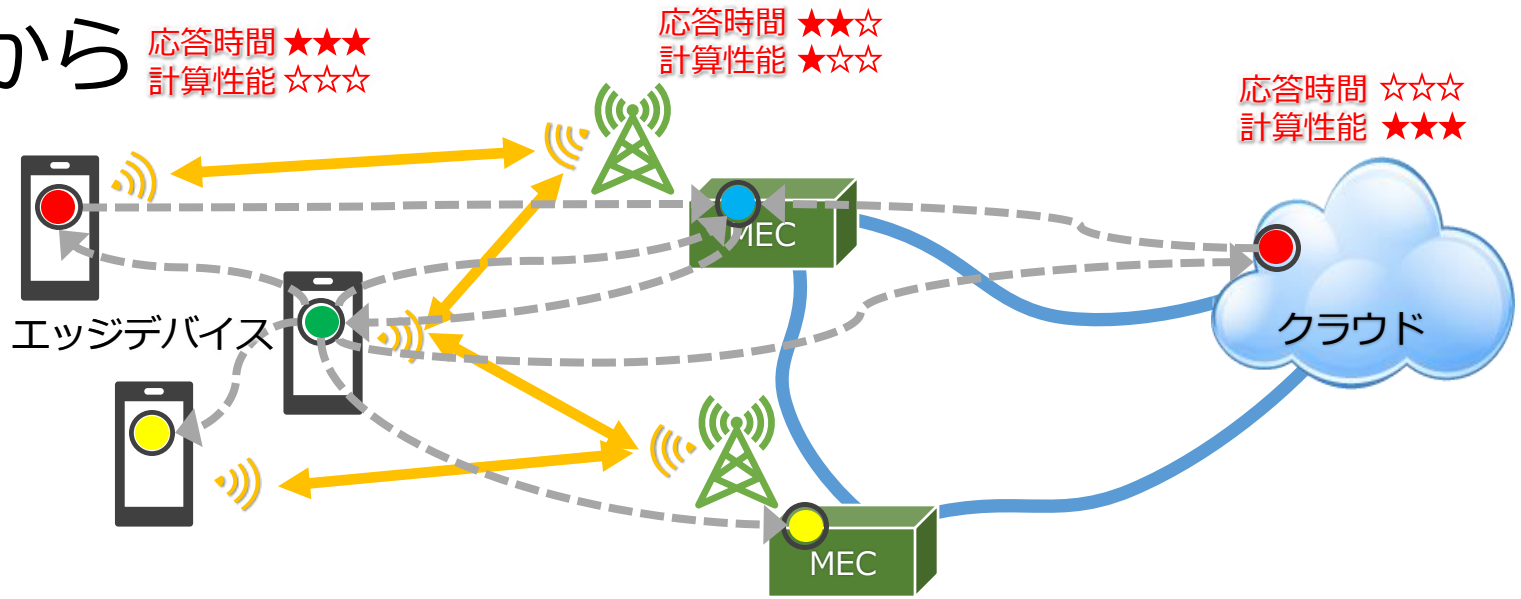


- 広域にまたがって分散してアプリ処理を実現する
  - 計算機間の通信とその運用がカギのひとつ
- 現代的にはデバイス／MEC（中間サーバ）／クラウドの構造をとる
  - 各々の計算資源はバラバラ：応答時間 vs 計算性能

広域分散処理の実現に向けた課題

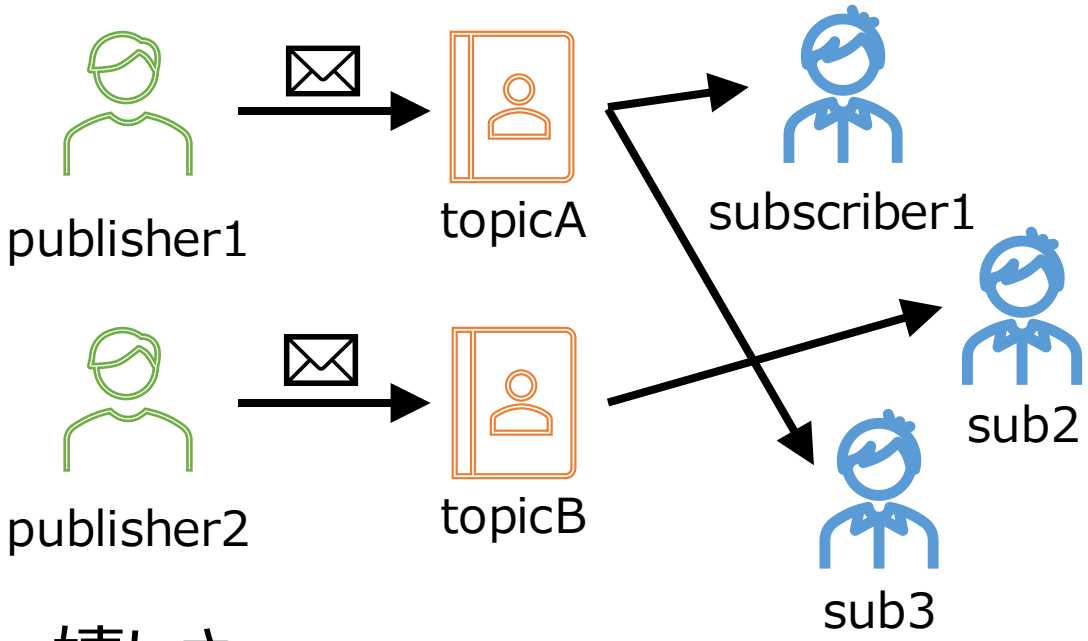
- システム開発者視点から





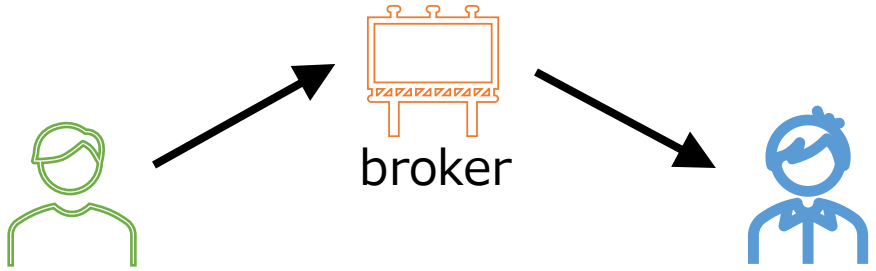
- なにをどこで処理するのか？
- **それらをどのように繋げるのか？**
- これをどうやって実現するのか？

出版購読型通信



- 嬉しさ
  - 非同期かつ疎結合なアーキテクチャを形成しやすい
  - それぞれのノードが独立して動作する追加／削除／再配置が容易

- Brokered (e.g., MQTT)
  - brokerがどこかを把握する必要アリ



- Peer-to-Peer (e.g., DDS)
  - 自律的に通信相手を探索できる
  - 通常の適用範囲は同一ネットワーク内のみ

https://zenoh.io/

Home Documentation Use Cases Community Adopters Media Blog

Zenoh

Zero Overhead Network Protocol.

Zenoh /zeno/ is a pub/sub/query protocol unifying data in motion, data at rest and computations. It elegantly blends traditional pub/sub with geo distributed storage, queries and computations, while retaining a level of time and space efficiency that is well beyond any of the mainstream stacks.

Learn more

Eloquent

First class abstractions for *pub/sub*, *geo-distributed storages*, *query*, and *queryables* simplify the development of distributed applications, at any scale.

Pub/Sub
Zenoh provides efficient publish/subscribe primitives,

Scalable

Zenoh protocol and its implementations are decentralized and can scale-out as well as scale-down.

Scalable Routing
Zenoh provides scale-out through a scalable routing infrastructure that allows your applications to be

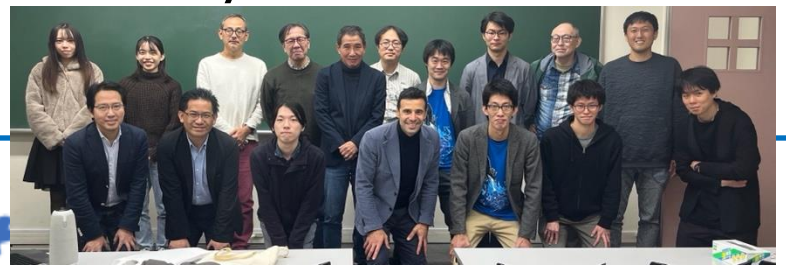
Fast

Zenoh is fast to learn and extremely performant.

Fast Adoption
Zenoh has a very simple API that makes it extremely fast to get started and productive. APIs are available for the most popular programming languages and more are

Zenoh

- Zero Overhead Pub/Sub, Store/Query and Compute.
  - **Ze**ro **n**etwork **o**ver**h**ead protocol
  - ネットワーク内ではDDSライク，
    ネットワーク間ではMQTTライクな通信ミドルウェア
  - GitHub：https://github.com/eclipse-zenoh/
    - ✓Eclipse Public License 2.0 and/or Apache 2.0
  - 当初はOCaml実装，2020/10- 頃からRustに移行
- 開発主体：ZettaScale Technology
  - CEO/CTO：Angelo Corsaro さん
  - ADLINK（CycloneDDS開発主体）から独立？



Zenoh

とにかくかるい！

いろいろつながる！

Zenoh

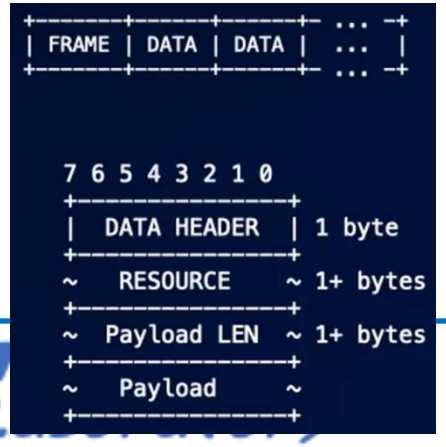
なんでもつかえる！

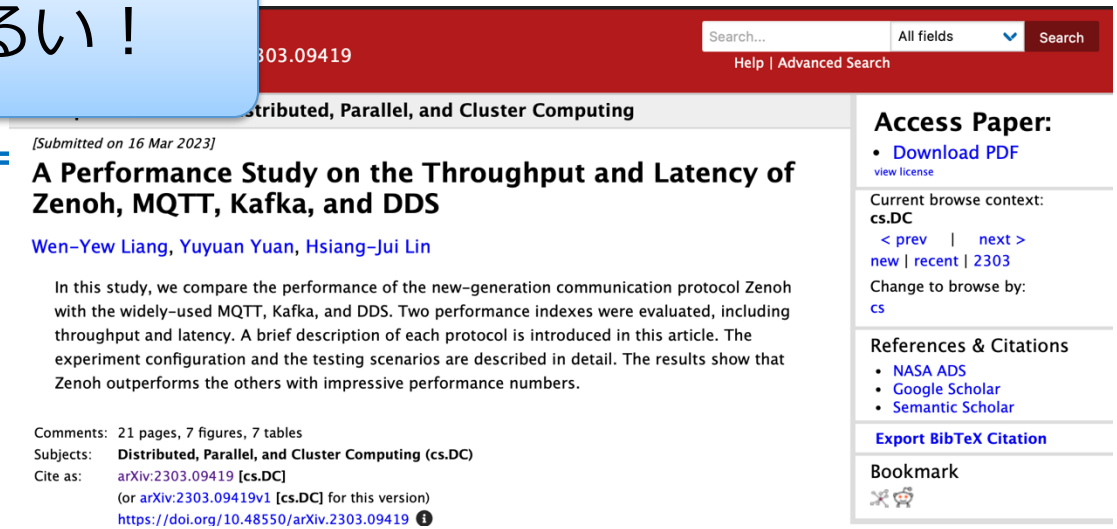
よしなにしゃべれる！

Zenoh

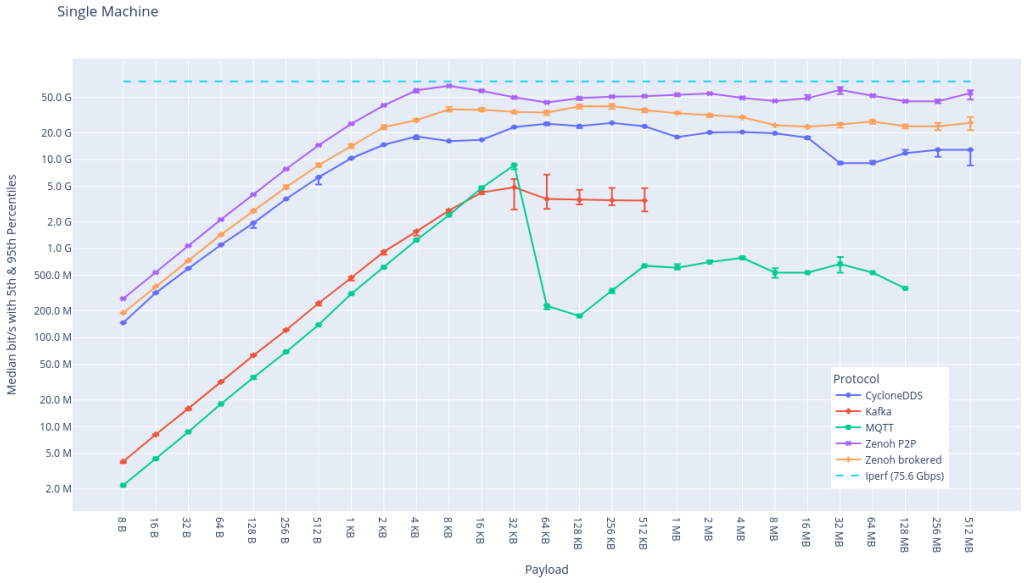
とにかくかるい！

- Low latency and High throughput
  - 10 us latency in the single machine,
    16 us in multiple machines (P2P config.)
  - ~70 Gbps at 8 KB payload
    - ✓35x higher than MQTT,
      23x than Kafka, 3.3x than DDS
- Why?: minimum wire overhead
  - only 5 bytes for delivering messages





arxiv:2303.09419

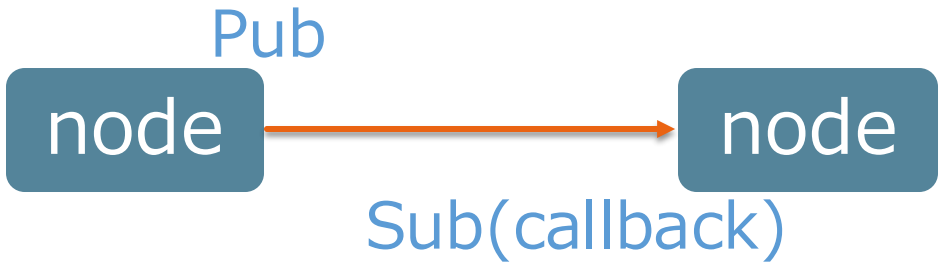


https://zenoh.io/blog/2023-03-21-zenoh-vs-mqtt-kafka-dds/

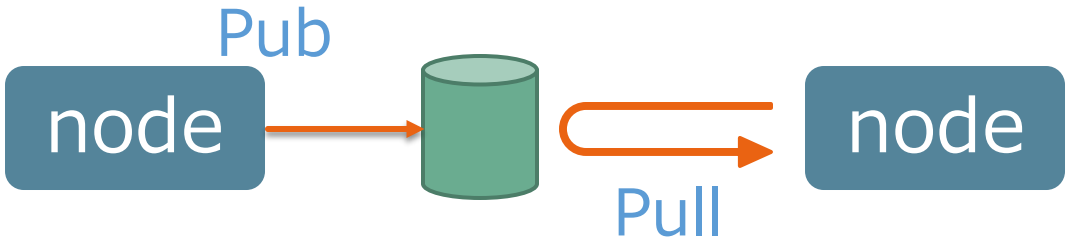
Zenoh

なんでもつかえる！

- Pub / Sub (Push)
  - basic pub/sub method



- Pub / Store / Get
  - KVS based computation

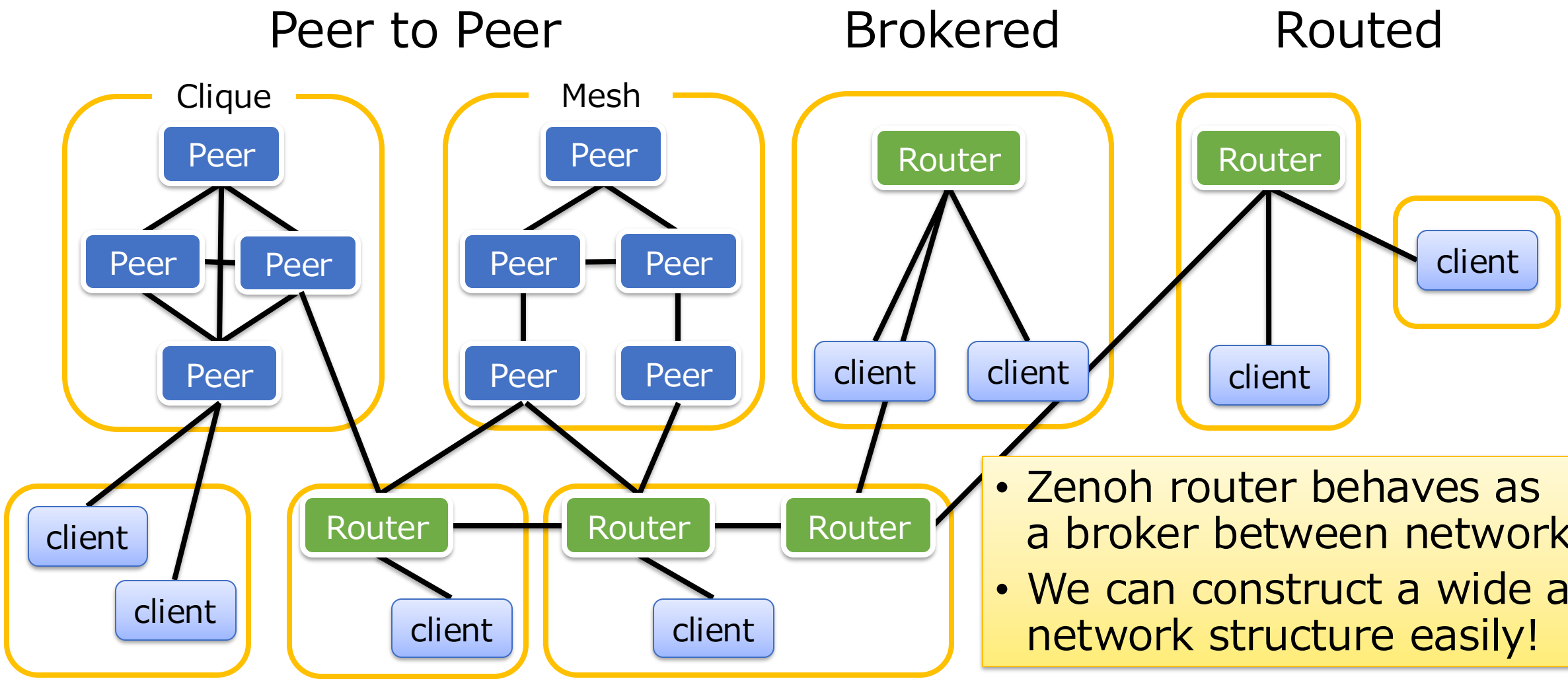


- Pub / Sub (Pull)
  - Sub receives in its own timing

- Get / Reply
  - RPC-like communication

Zenoh

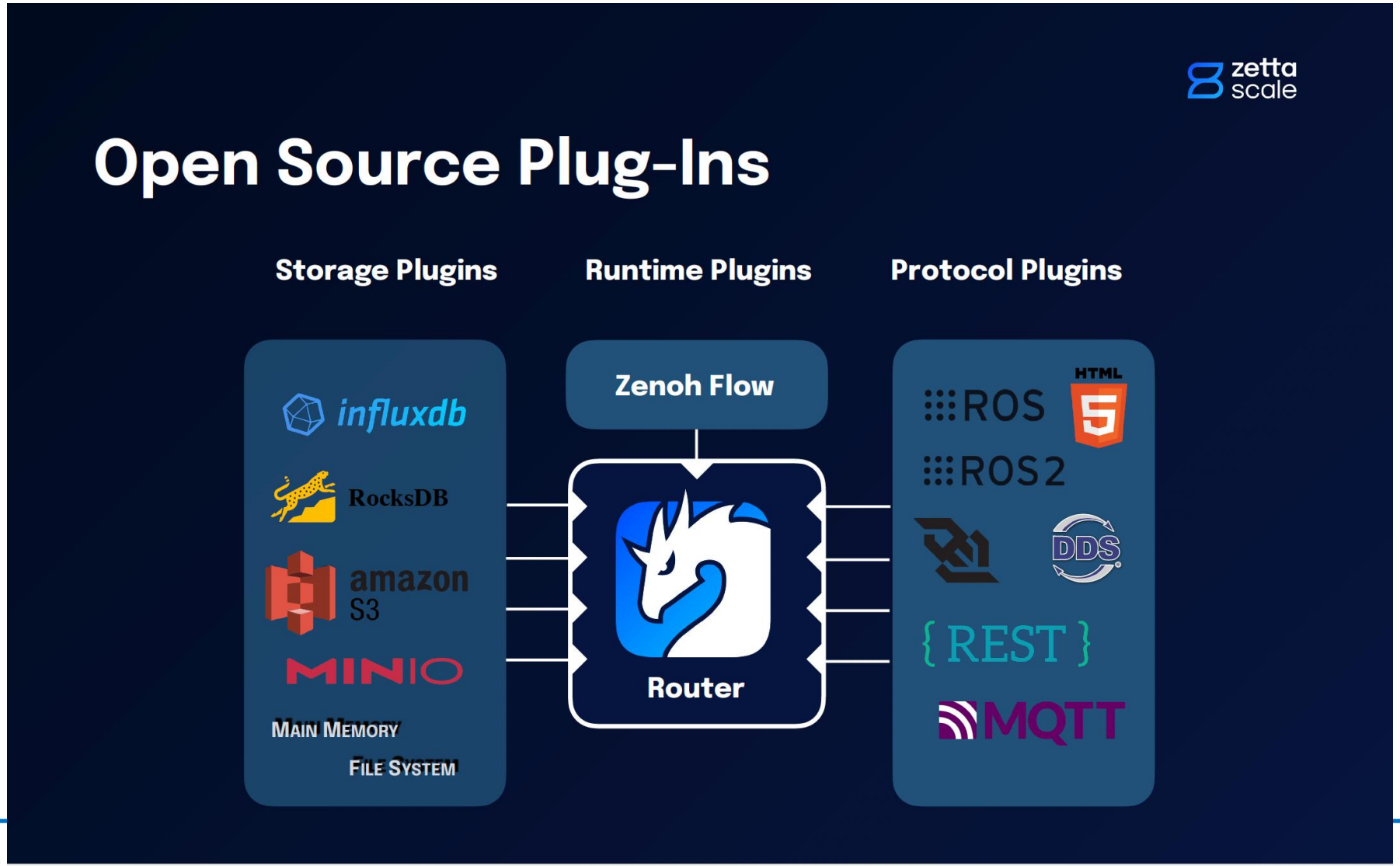
いろいろつながる！



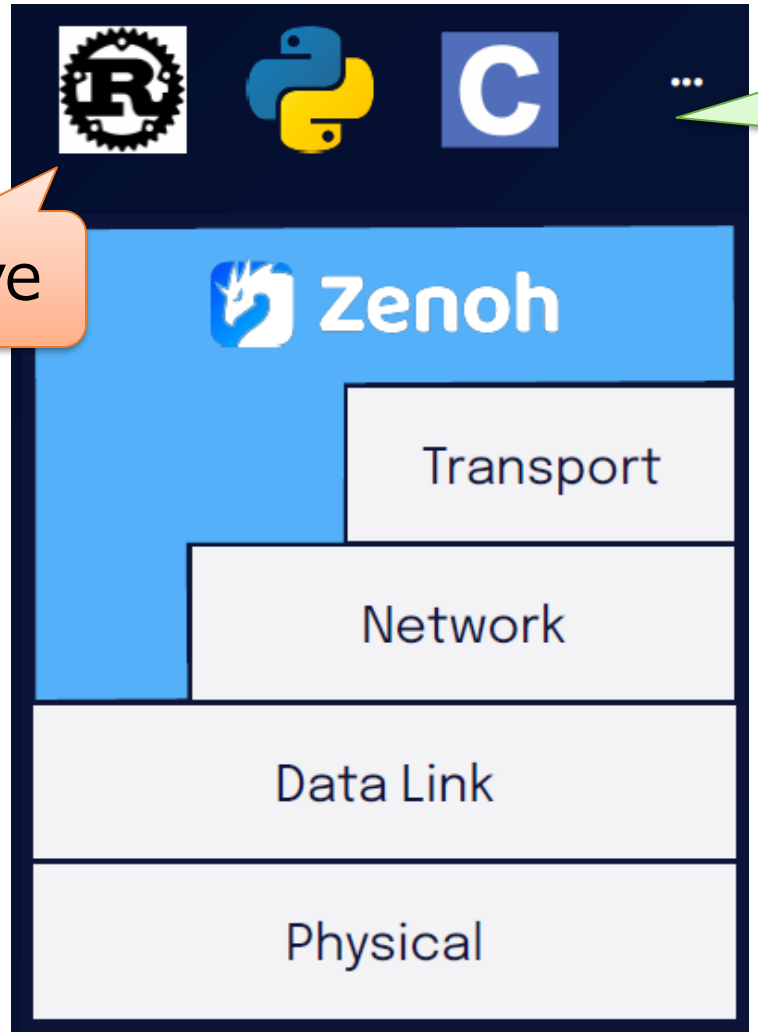
- Zenoh router behaves as a broker between networks
- We can construct a wide area network structure easily!

Zenoh

よしなにしゃべれる！



Zenoh Runs Everywhere!



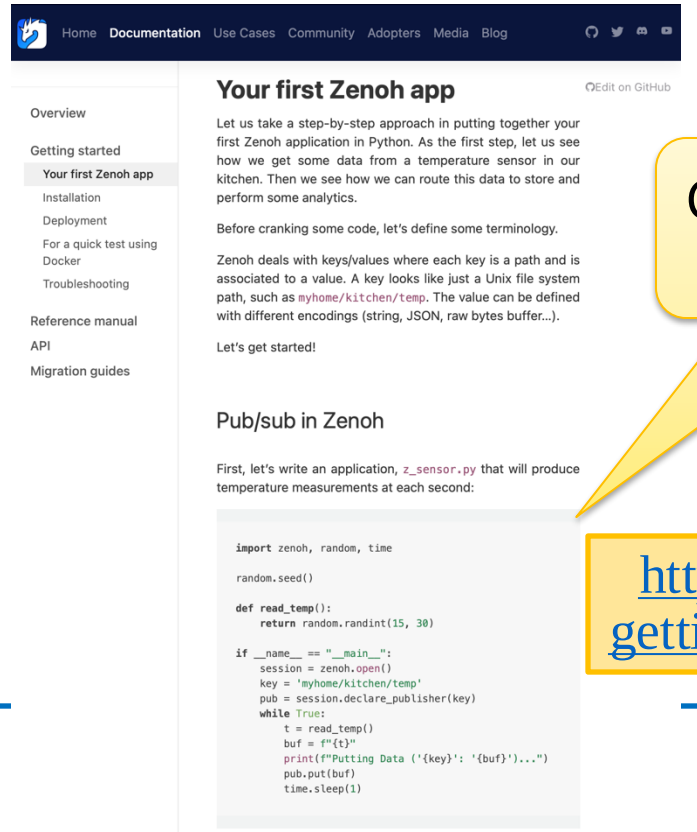
APIs for various languages

- zenoh-python
- zenoh-kotlin
- zenoh-c
- zenoh-csharp
- zenoh-cpp
- zenoh-go
- zenoh-java

QUIC, TLS, TCP,
UDP Unicast,
UDP Multicast

IPv4, IPv6
6LoWPAN

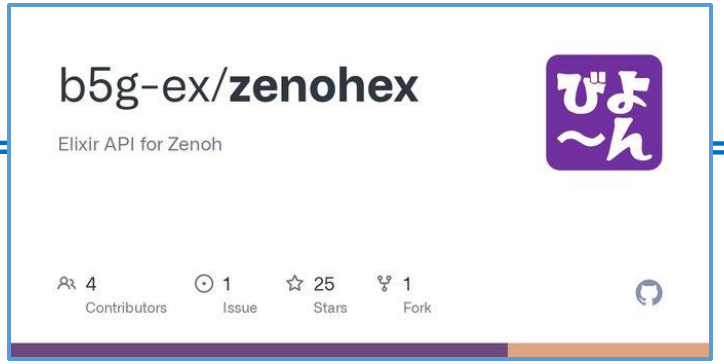
WiFi, Ethernet,
Bluetooth, Serial



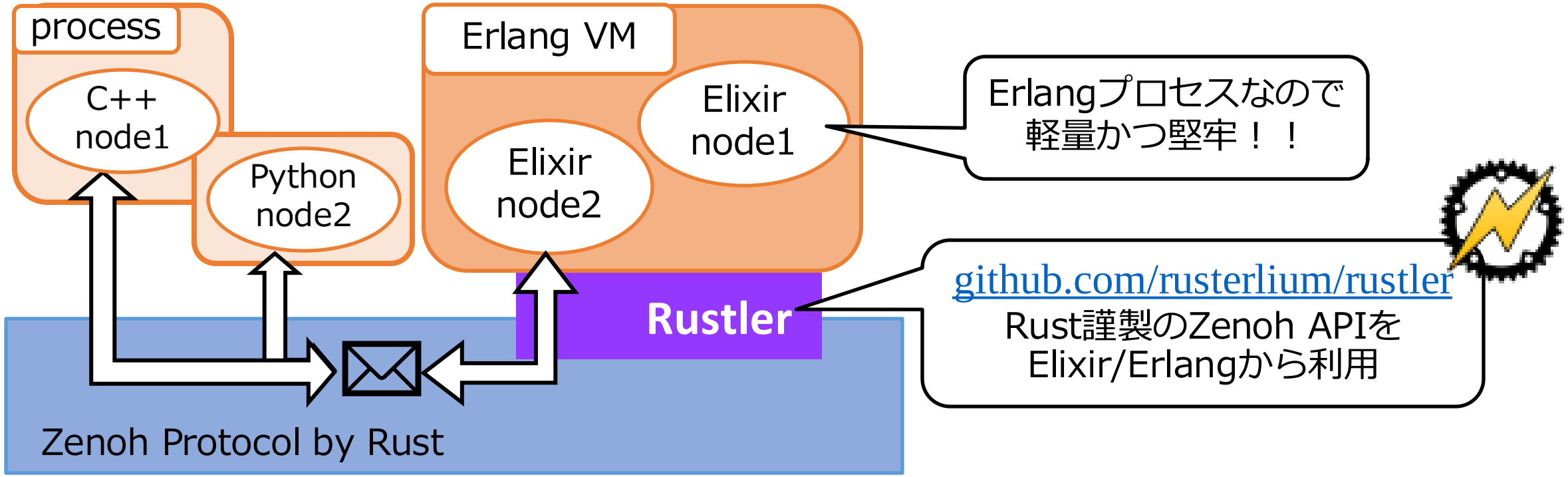
Getting Started with Python

https://zenoh.io/docs/getting-started/first-app/

だったらElixirじゃね？？



- **Zenohex = Zenoh + Elixir**
  https://github.com/b5g-ex/zenohex



なんしか Elixir!!

elixir

2012年に登場した新しめの関数型言語

BEAM (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
- 通信応答性能が極めて高い
- **Web/IoT/AI FW. を備える**

NERVES

Elixir *Zen* Style

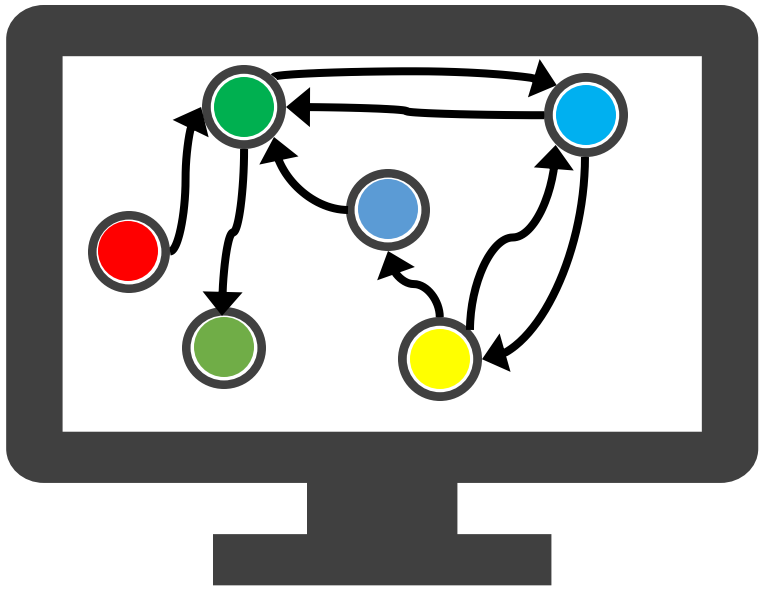
```
1..1000
  |> Flow.from_enumerable()
  |> Flow.map(& foo(&1))
  |> Flow.map(& bar(&1))
  |> Enum.to_list
  |> Enum.sort
```

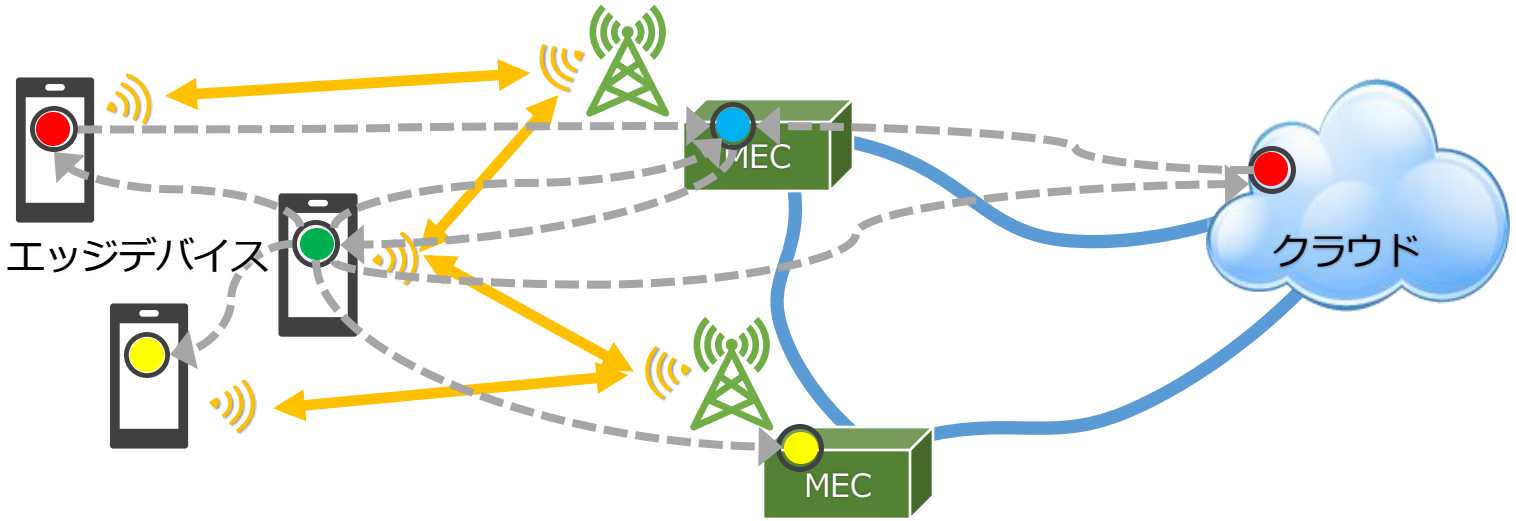
Programming should be about transforming data

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

※各言語・フレームワークのロゴの権利は
それぞれの作成者に帰属しています

関数型パラダイム ＝ IoT Computing!!

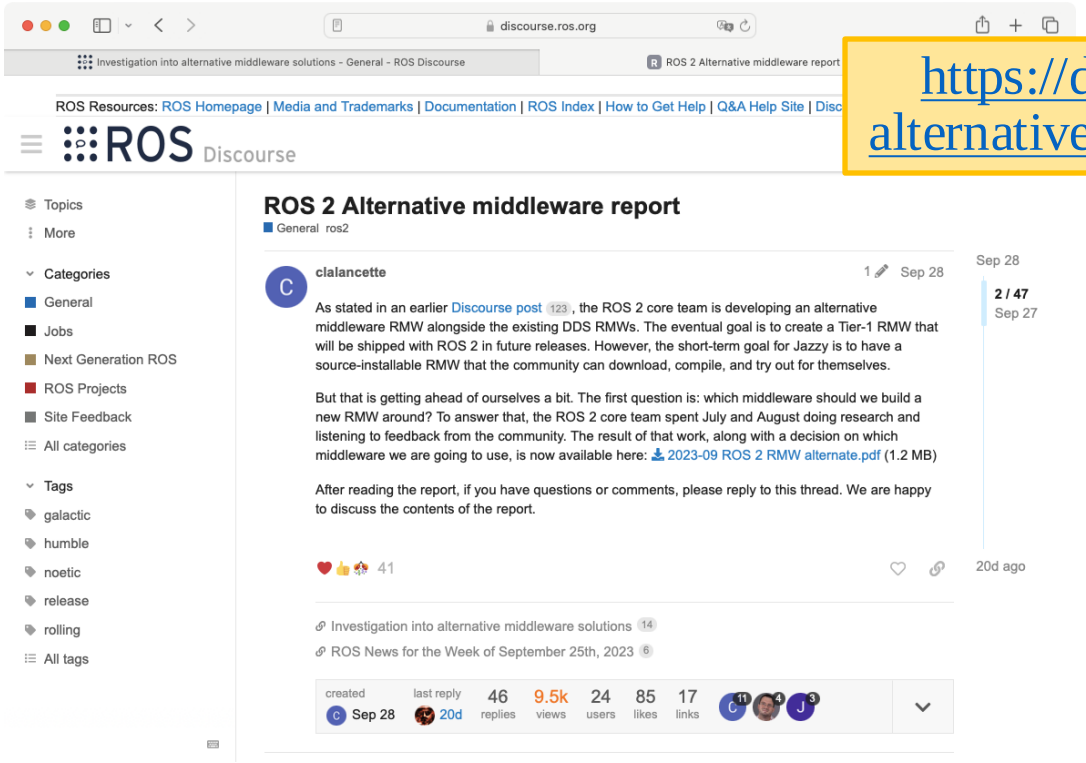




プロセス同士の通信 ＝ Things同士の通信

計算機同士でクラスタを形成すれば
プロセス同士が相互に作用して動作する！

そしてROS界隈では,,,

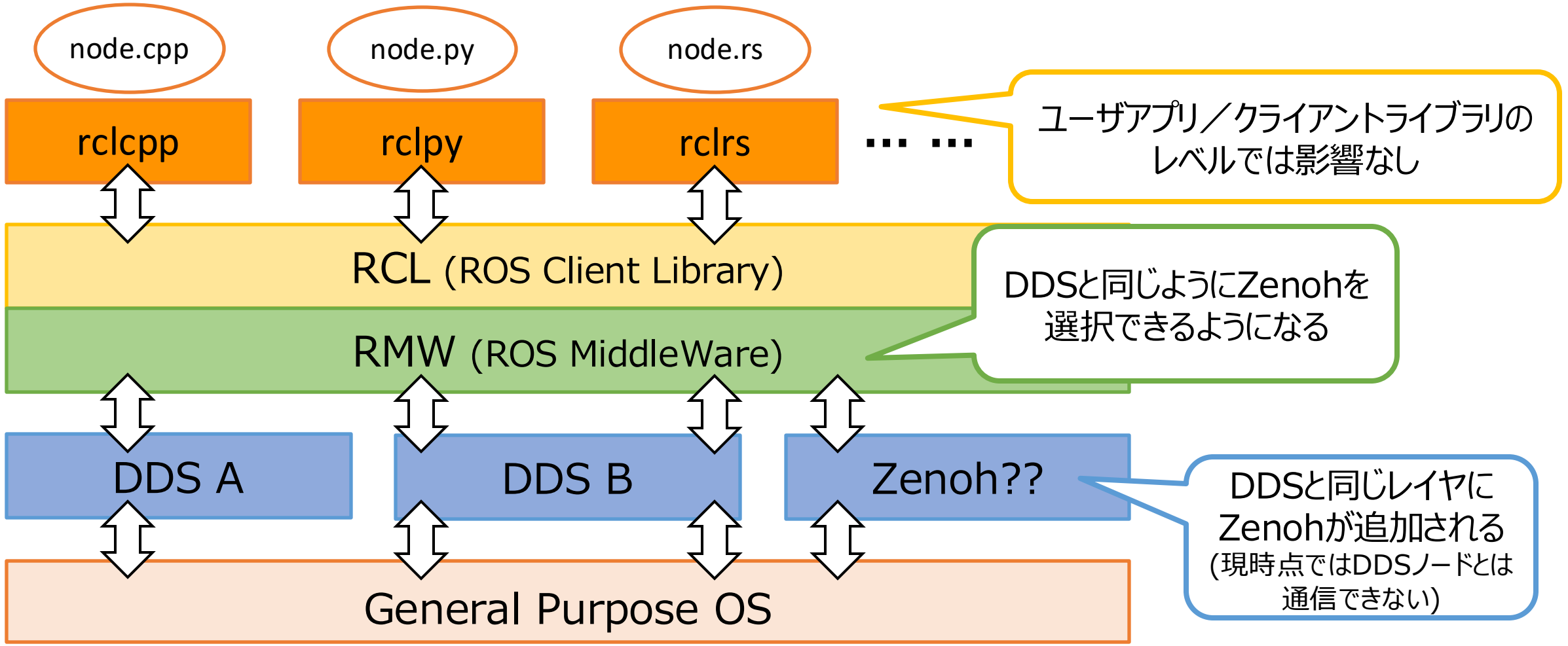


https://discourse.ros.org/t/ros-2-alternative-middleware-report/33771

Conclusion

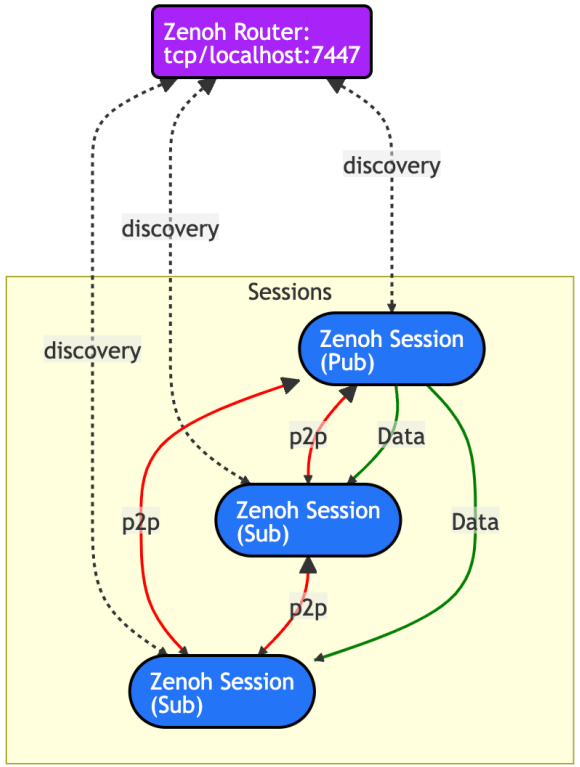
Requirements from ROS 2 users were gathered, and middleware options that are available today were investigated. The research has concluded that Zenoh best meets the requirements, and will be chosen as an alternative middleware. Zenoh was also the most-recommended alternative by users. It can be viewed as a modern version of the TCPROS implementation, and meets most of the ROS 2 requirements. There are still a number of design decisions to be made regarding this implementation; those details will be discussed on https://discourse.ros.org as development begins.

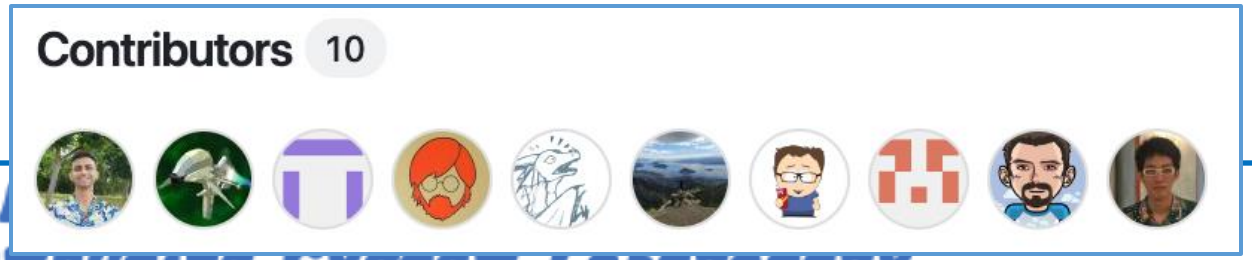
つまり，こうなる！(はず)


node.cpp
node.py
node.rs
rclcpp
rclpy
rclrs
… …
ユーザアプリ／クライアントライブラリの
レベルでは影響なし
RCL (ROS Client Library)
RMW (ROS MiddleWare)
DDSと同じようにZenohを
選択できるようになる
DDS A
DDS B
Zenoh??
DDSと同じレイヤに
Zenohが追加される
(現時点ではDDSノードとは
通信できない)
General Purpose OS

ros2/rmw_zenoh

- **ROS 2**が**Zenoh**でつながる
  - zenoh-c (C API for Zenoh) の binding として実装
    階層：RMW(C++) |> API(C) |> Zenoh(Rust)
  - ROS 2 node = Zenoh Session
    Zenoh Router で discovery |> P2P で pub/sub
    - ✓現状は Router の立ち上げが必須（将来的には不要に？）
  - maintainers:
    - ✓Yadunund Vijay (@Yadunund Iron ROS Boss)
    - ✓Chris Lalancette (@clalancette ROS 2 Technical Lead)



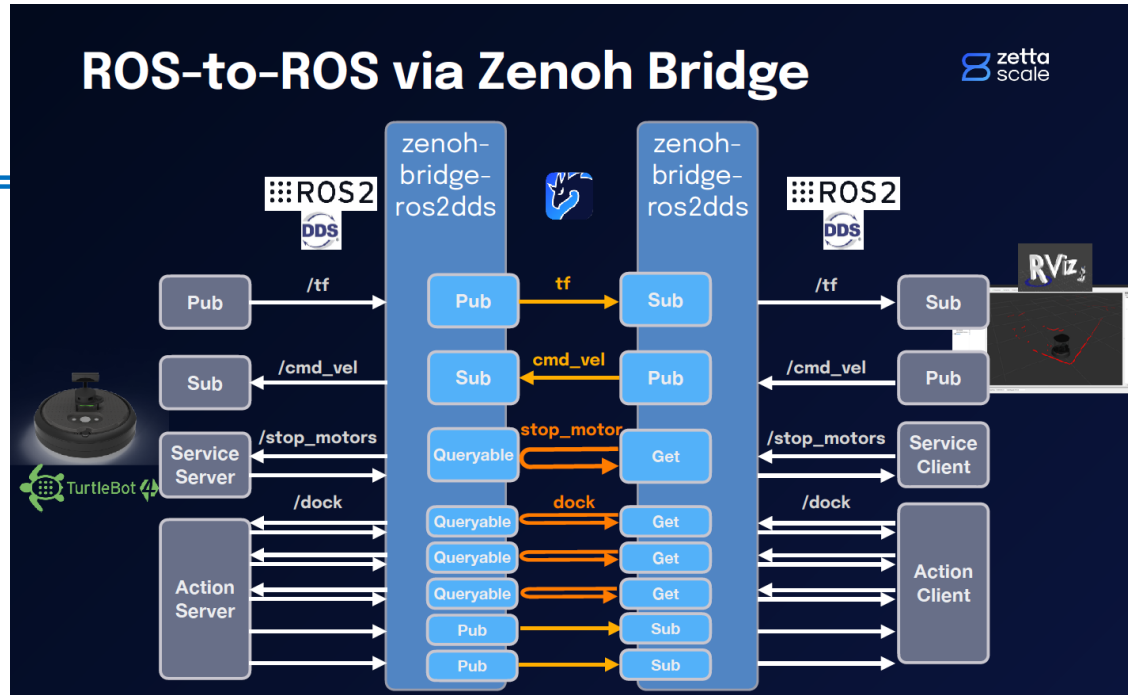


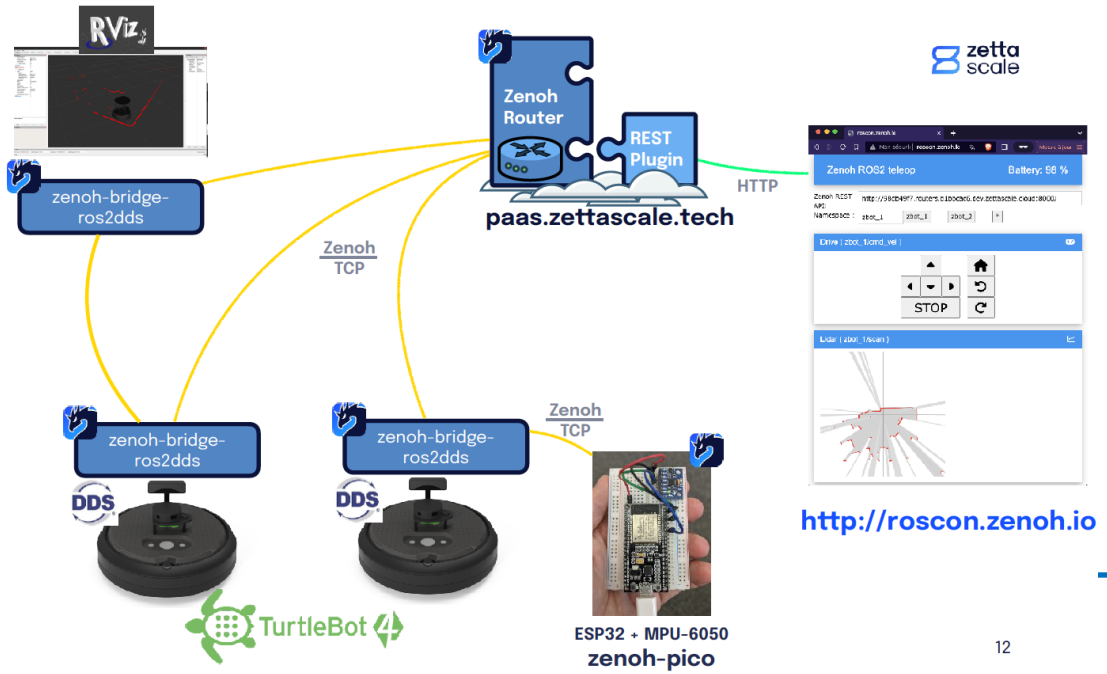
そしてさらに！

New Zenoh Bridge for ROS 2

https://github.com/eclipse-zenoh/zenoh-plugin-ros2dds

- Better integration of the **ROS graph**
  (all ROS topics/services/actions can be seen across bridges)
- Better support of **ROS toolings**
  (ros2, rviz2…)
- Configuration of a **ROS namespace** on the bridge
  (instead of on each ROS Nodes)
- Services and Action as **Zenoh Queryables**
- Even **more compact discovery information** between the bridges





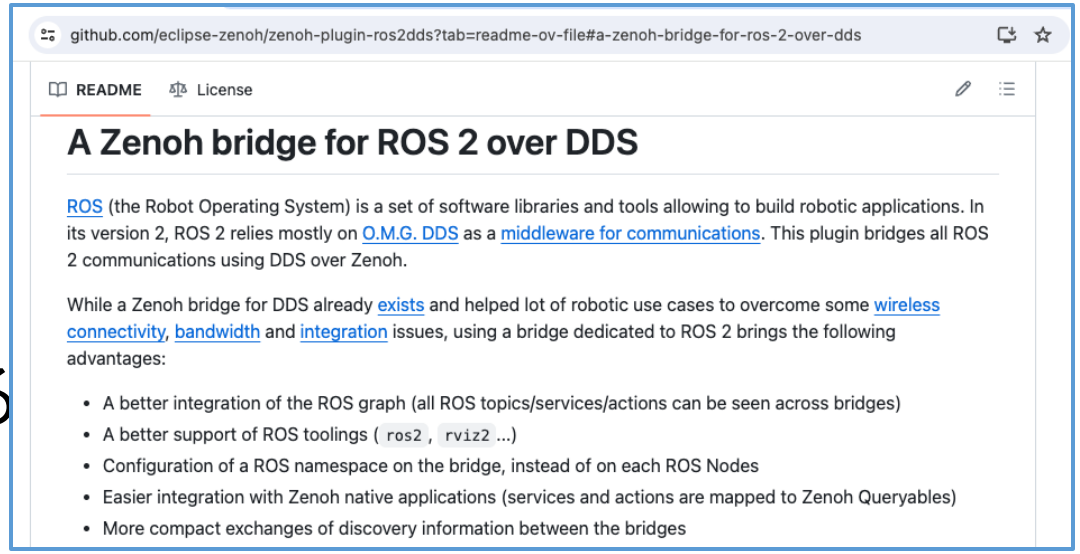
ROSJP#53「ROSCon 2023参加報告：Real-Time Workshop & Zenoh's Current Status and Forecast」

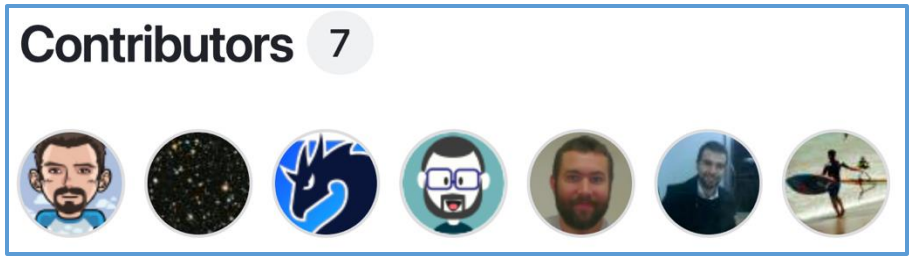
https://speakerdeck.com/takasehideki/roscon-2023can-jia-bao-gao-real-time-workshop-and-zenohs-current-status-and-forecast

https://www.youtube.com/watch?v=TZaYVL8xeBs&t=2426s

zenoh-plugin-ros2dds

- **Zenoh**で**ROS 2**がつながる
  - zenoh-plugin-dds から派生
  - より ROS 2 フレンドリーに！
  - plugin: router にロードして使う
  - bridge: スタンドアロンで動作する
  - maintainers:
    - ✓Julien Enoch (@JEnoch
      Senior Solutions Architect at ZettaScale)





本題！組込みマイコンでも！！

- https://github.com/eclipse-zenoh/zenoh-pico
  - Cで再実装した組込み向け軽量Zenoh
  - PlatformIO で各種PF/RTOS/ボード向けに対応
  - かなりええ感じの性能が出ているらしい
    https://zenoh.io/blog/2022-02-08-dragonbot/

In this post, we will dive deeper on Zenoh-Pico, show, how Zenoh-Pico is capable of:

- exchanging close to **2.5M msg/s** for small payloads, and over **25 Gbps** for larger messages,
- achieving end-to-end latency (i.e., one way delay) as small as **45 µsec** and **15 µsec** for unicast and multicast transports, respectively,
- minimizing the overhead in the wire down to **5 bytes** per data transmission,
- fitting all its capabilities in less than **50KB** footprint, which can be quickly reduced to **~15KB** in tailored compilation setups, and
- provides **simple** to use and yet **powerful APIs**.

(RT)OS	Transport Layer	Network Layer	Data Link Layer
Unix	UDP (unicast and multicast), TCP	IPv4, IPv6, 6LoWPAN	WiFi, Ethernet, Thread
Zephyr	UDP (unicast and multicast), TCP	IPv4, IPv6, 6LoWPAN	WiFi, Ethernet, Thread
Arduino	UDP (unicast and multicast), TCP	IPv4, IPv6	WiFi, Ethernet, Bluetooth (Serial profile)
ESP-IDF	UDP (unicast and multicast), TCP	IPv4, IPv6	WiFi, Ethernet
MbedOS	UDP (unicast and multicast), TCP	IPv4, IPv6	WiFi, Ethernet, Serial
OpenCR	UDP (unicast and multicast), TCP	IPv4	WiFi

論よりRUN!!

"ron yori run"
The RUN is mightier than the word

- ハンズオン１：様々な言語でつなげる
  - Python, Elixir, Rust 実装のノードをPub/Subしてみる
  - PhoenixアプリとWeb連携させてみる
  - プログラミングモデルを理解する
- ハンズオン２：IoT的につなげる
  - zenoh-pico を使ってみる
  - 組込みマイコンとWebアプリをIoT連携させてみる
- 詳細な手順は,,,
  https://github.com/takasehideki/zenoh_swest26_trial
  - 次ページ以降は要点のみを示しています

ハンズオン１：様々な言語でつなげる

- 下準備：コンテナの起動

```
cd zenoh_swest26_trial
docker compose up -d
```

- ターミナルを８つ開いて Docker に入る（ペインで２行４列がオススメ）

```
cd zenoh_swest26_trial
docker compose exec app bash
```

- ノードやアプリをビルドする

```
cd zenoh_python
```

```
cd zenoh_python
```

```
cd zenoh_elixir
mix deps.get && mix compile
```

```
cd zenoh_elixir
```

```
cd zenoh_native
cargo build
```

```
cd zenoh_native
```

```
cd zenohex_phoenix_demo
mix setup && mix compile
```

ハンズオン１：様々な言語でつなげる

- Let’s talk!!

```
python3 sub.py
```

```
python3 pub.py
```

```
iex -S mix
iex()> ZenohElixir.Sub.main()
```

```
iex -S mix
iex()> ZenohElixir.Pub.main()
```

```
./target/debug/sub
```

```
./target/debug/pub
```

```
mix phx.server
```

- ソースコードを眺めてみましょう
- key を変えたりいろいろ試してみましょう

プログラミングモデル

- Resource:
  - (key, value)ペアの名前付けされたデータ
  - 例：home/kitchen/sensor/temp, 21.5
    home/kitchen/sensor/hum, 0.67
- Key expression:
  - keyの集合表現（ワイルドカード？）
  - 例：home/kitchen/sensor/*
    home/**/temp
- Selector:
  - resource 集合を特定する表現
  - 例：home/*/sensor/air?_where=co2>12&_project=humidity
- open/close:
  - zenohセッションを開始／終了する
  - 引数等でネットワーク構成を指定する
    - ✓ 多くの場合では1つのセッションを使い回すことが推奨されている
- declare_keyexpr:
  - key-expressions を宣言する
- declare_publisher:
  - 出版者として宣言する
- declare_subscriber:
  - 購読者として宣言する
  - 購読時に実行されるコールバック関数を指定する

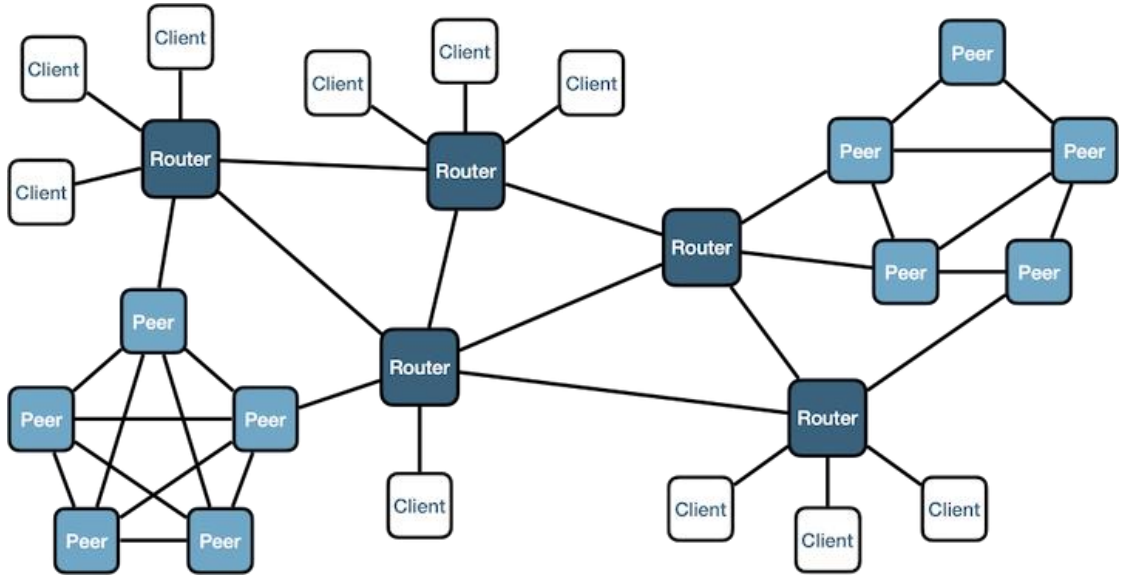
ハンズオン２：IoT的につなげる

- 下準備
  - `zenoh_d3ai_trial/zenoh_pico/` のディレクトリを PlatformIO Project として開く
  - `src/main.cpp` を編集する
    - ✓ 13-14行目の `SSID` と `PASS` を本日のWi-Fi AP環境に合わせる
    - ✓ 19行目の `CONNECT` を自身のPCのIPとポート番号に合わせる
      #define CONNECT "tcp/192.168.x.x:7447"
  - Build, Upload, and RUN!!!
- いろいろ試してみてください

- 補足：
  - がむばれば他のM5ボードでもできますが platform.ini の記述がいろいろ必要なことがあります（こんな感じ）
  - zenohd (Zenoh router) は IP reachable であれば接続できます　でも今回はちょっとサボっています,,, (こんな感じ)
  - がむばれば Cloud VM とも連携可能です（こんな感じ）

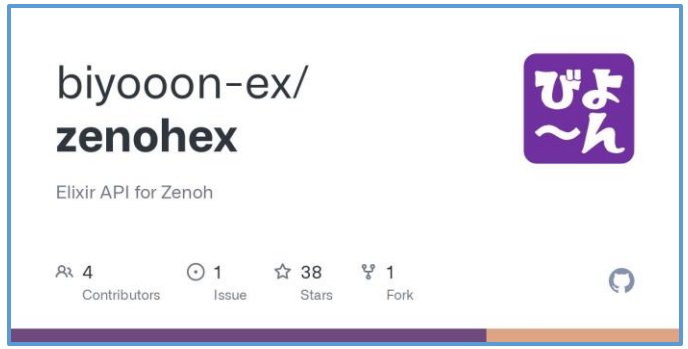
ネットワークモデル

- Peer:
  - Zenohアプリとしての基本
  - ローカルネットワーク内では通信相手を自律的に探索できる（like DDS）
  - Scouting: 通信相手の探索方式の指定（multicast と gossip の方式がある）
- Client:
  - ルータを介して通信する構成
- Zenoh router:
  - 通信を仲介するモジュール
  - IP or URL, TCP or UDP で指定可能
  - ポート番号も指定可能（基本は7447）



JSONとかでいろいろ指定できるらしい,,,
https://zenoh.io/docs/getting-started/deployment/

Thank you!
merci!!
Arigato!!!



Zenoh

@takasehideki

elixir

A part of this work is going as collaborative research with SoftBank Corp., and was supported by the commissioned research (04001) by National Institute of Information and Communications Technology (NICT), Japan.