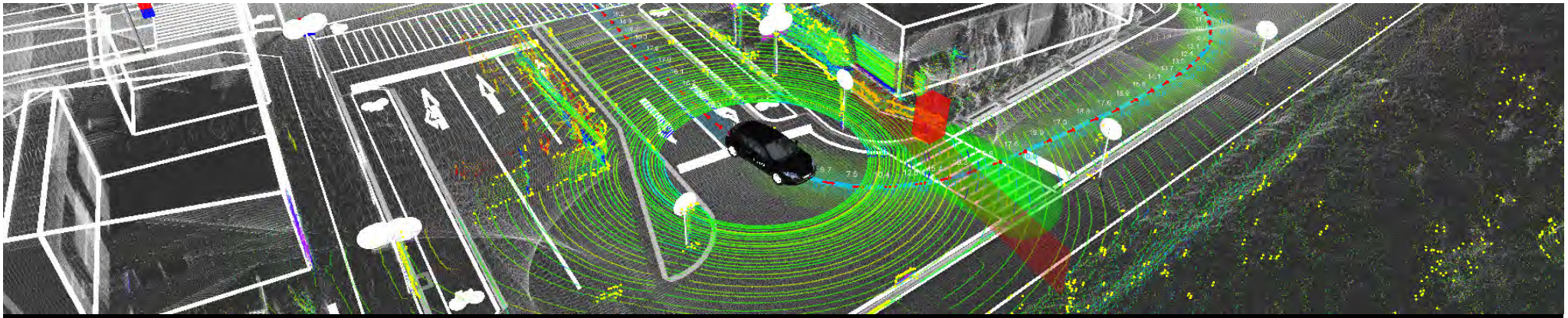


自動運転プラットフォームの実用化：ROS 2で高信頼ソフトウェアの実装

安積卓也

Geoffrey Biggs

AutoWare

Saitama University 埼玉大学

EmbIV Intelligent Vehicle

Tier IV Intelligent Vehicle

目次

- 自動運転ソフトウェアAutoware（Autoware.AI）
  - 自動運転レベル
  - ROS
  - 自己位置推定
  - 環境認識
  - 経路計画・経路追従

実証実験で使われている
自動運転プラットフォーム
２５分程度

- 自動運転ソフトウェア：ROS 2への移行理由
  - ROSの弱点
  - ROS2
  - 今あるAutowareとその問題
  - Autoware.AIとAutoware.Auto

Geoffrey Biggs

目次

- 自動運転ソフトウェアAutoware（Autoware.AI）
  - 自動運転レベル
  - ROS
  - 自己位置推定
  - 環境認識
  - 経路計画・経路追従

後半：残りの時間

- **自動運転ソフトウェア：ROS 2への移行理由**
  - ROSの弱点
  - ROS2
  - 今あるAutowareとその問題
  - Autoware.AIとAutoware.Auto

Geoffrey Biggs

自己紹介

名前：安積　卓也（あづみ　たくや）

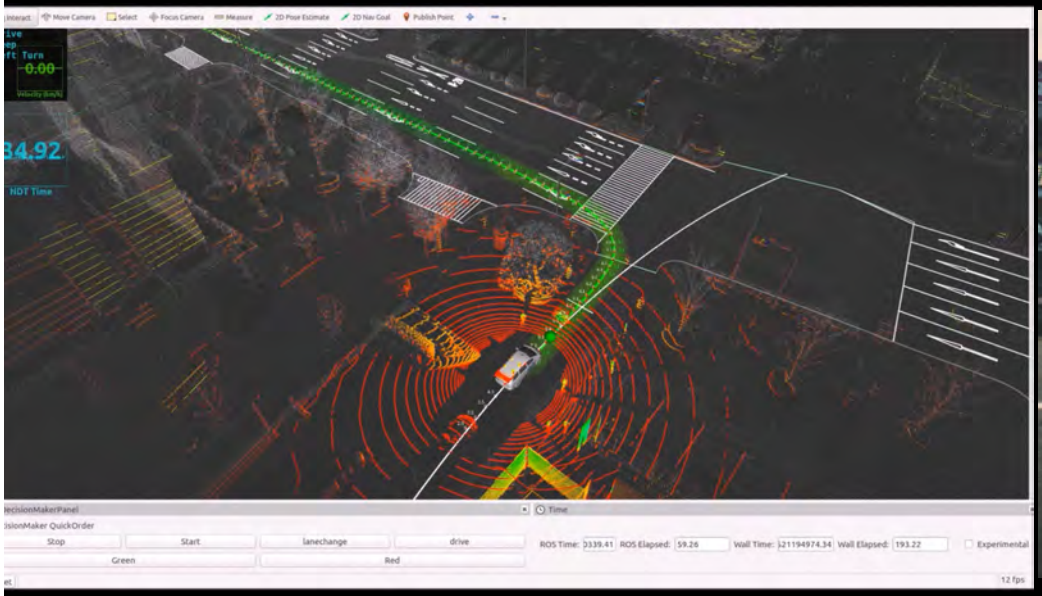
- 出身研究室：名古屋大学大学院　情報科学研究科
  - 2006–2009：博士後期課程 高田研：組込みリアルタイム
    2006-2007：未踏ソフトウェア（代表）
  - 2008-2010：学振　特別研究員　DC→PD
- 立命館大学情報理工学部・情報システム学科・助教
  - 2010年4月～2014年2月：西尾研究室：ユビキタス
  - 2011年9月～2012年3月：
    カリフォルニア大学アーバイン校：客員研究員

帰国後：自動運転向けシステムソフトウェアの研究開発開始

- 大阪大学 大学院基礎工学研究科 助教
  - 2014年3月～：潮研究室：制御・人工知能
  - 2016年9月～：株式会社エンブフォー：最高技術責任者
  - 2017年10月～：JSTさきがけ研究員（兼任）
- 埼玉大学大学院理工学研究科
  - 2018年4月～：准教授

EmbIV
Intelligent Vehicle

Autowareを用いた公道実験・利用の例



自動運転のレベル

※１官民ITS構想RM用語対応表
操舵：ハンドル　（ステアリング）
加速：アクセル
制動：ブレーキ

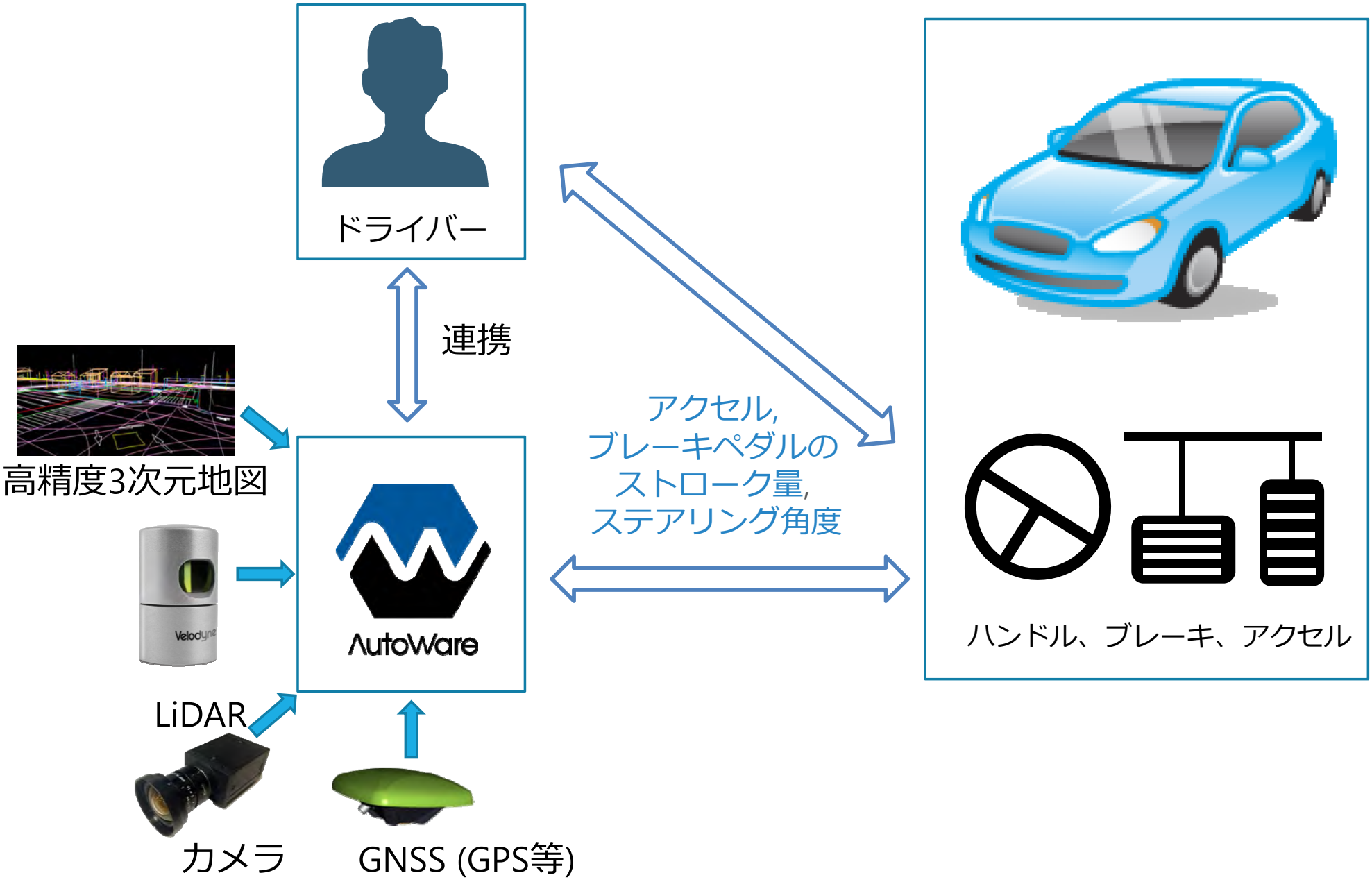
レベル	システム：※１ ハンドル・アクセル・ブレーキ	ドライバー	場所	
レベル１	いずれか一つ	主体		自動ブレーキ
レベル２	複数	主体		クルーズ コントロール
レベル３	すべて （危険回避はドライバー）	あり		公道実験
レベル４	すべて	なし	限定	自動運転バス
レベル５	すべて	なし	全て	自動運転 タクシー

完全自動運転

※SAE J3016（2016）

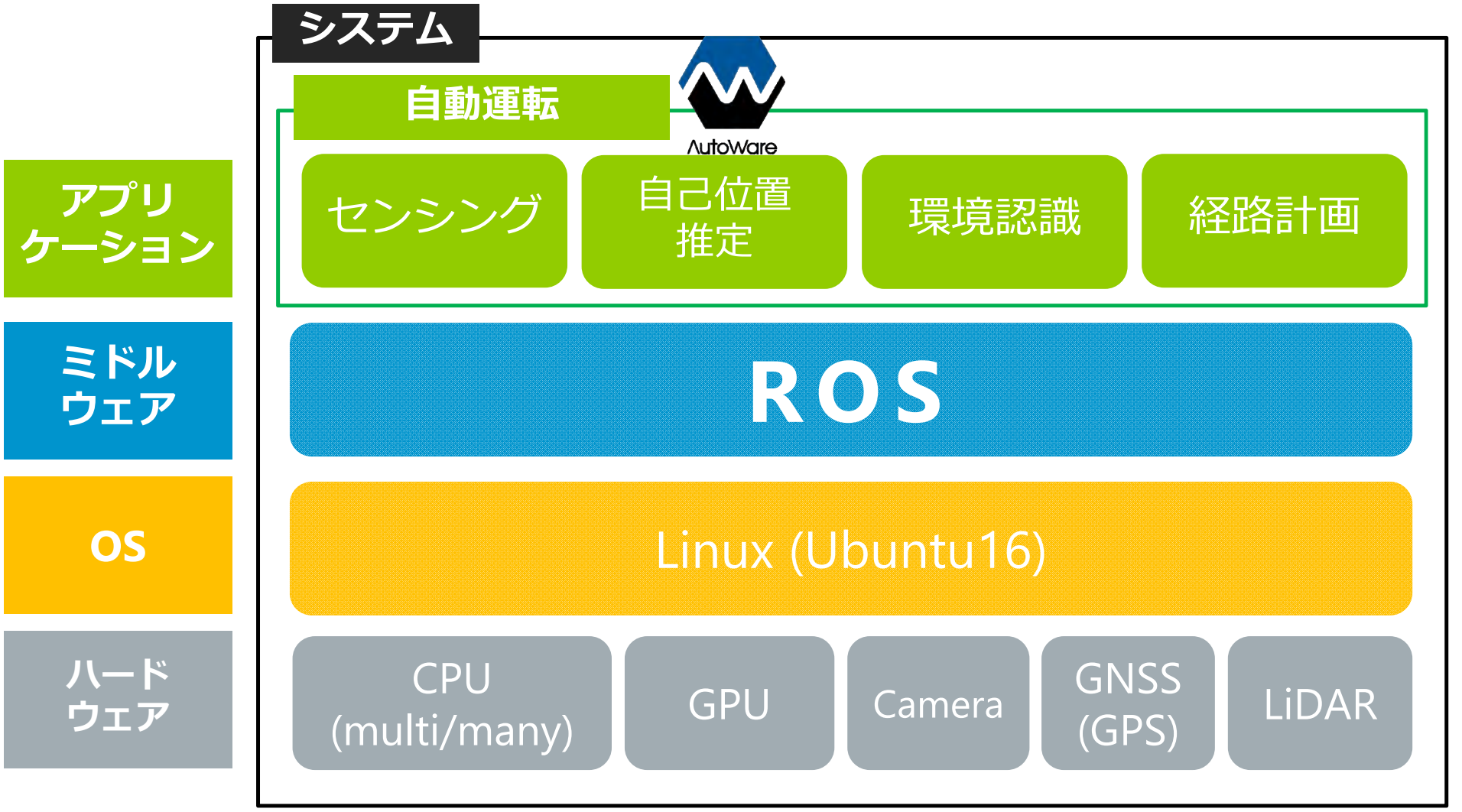
AutoWare

自動運転ソフトウェアの役目



https://gitlab.com/autowarefoundation/autoware.ai

Autowareの構成



https://gitlab.com/autowarefoundation/autoware.ai

ROSとは

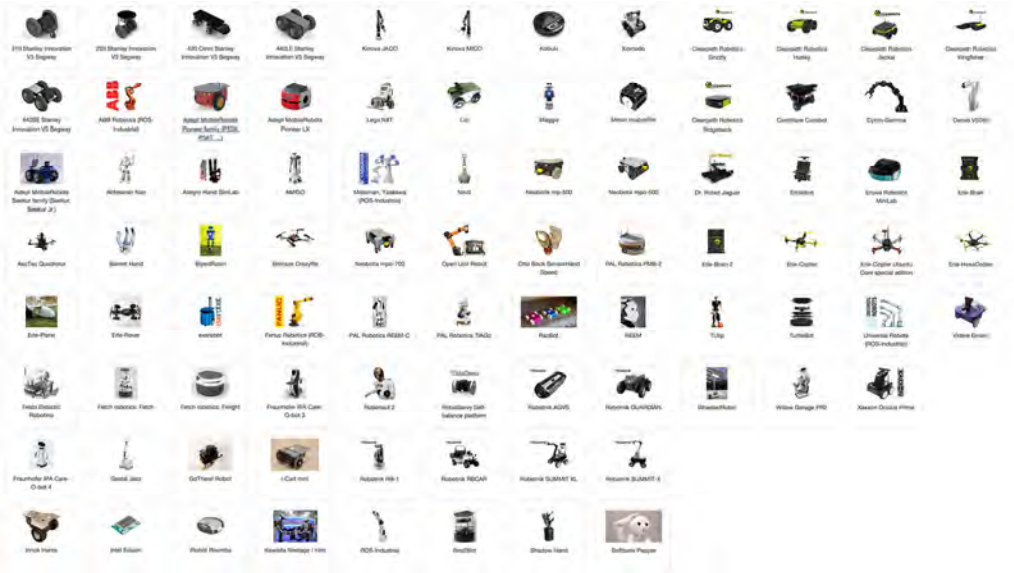
ROS (Robot Operating System)

: ロボット開発におけるライブラリやツールを提供

ハードウェアの抽象化、デバイスドライバ、ライブラリ、視覚化ツール、
データ通信、パッケージ管理 ...etc

特長

- 世界で最も利用されているロボットミドルウェア
- 豊富な対応ロボット・センサ
- オープンソース
- サポート言語： C++, Python
- 管理団体：OSRF
- 対応OS：Linux



ubuntu

Open Source Robotics Foundation

ROS の 特長

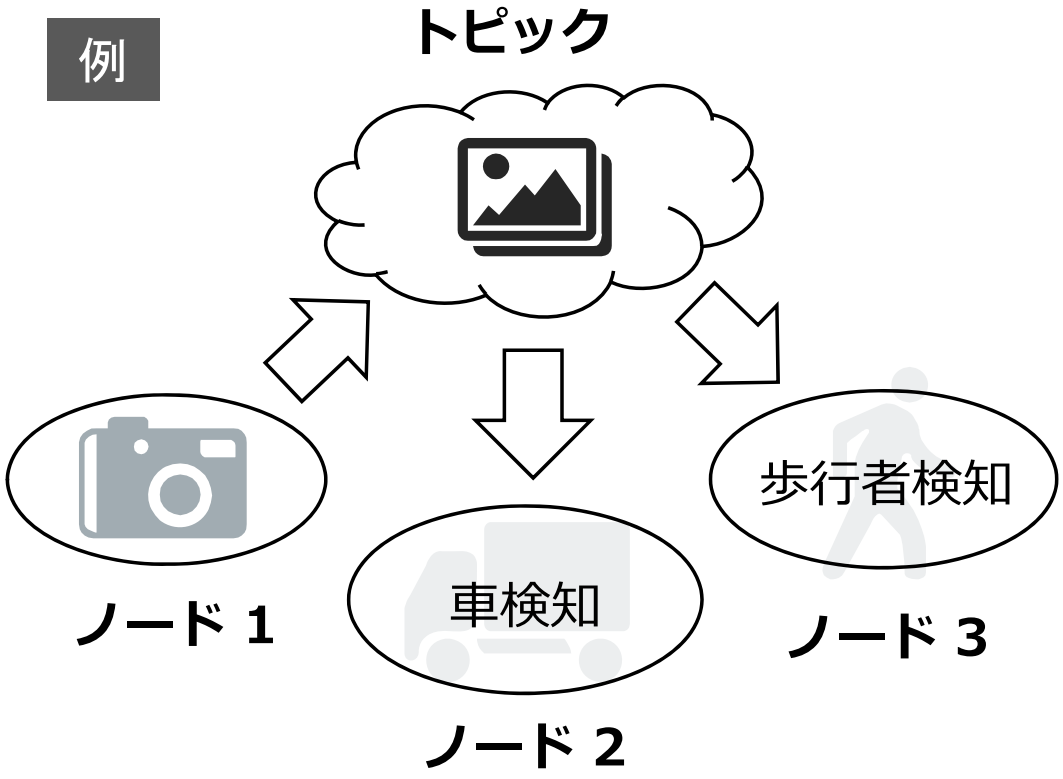
ROS (Robot Operating System)

：ロボット開発におけるライブラリやツールを提供

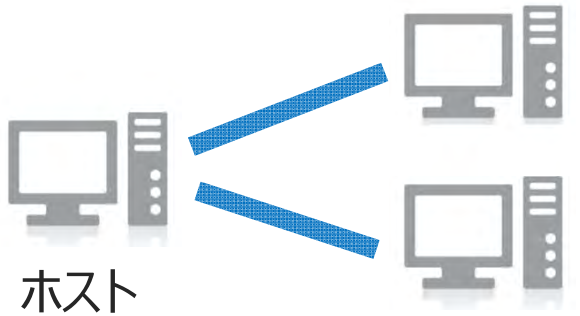
Publish / Subscribe モデル

- **ノード**の集合としてシステムを構築
- **トピック**を介してデータをやり取り

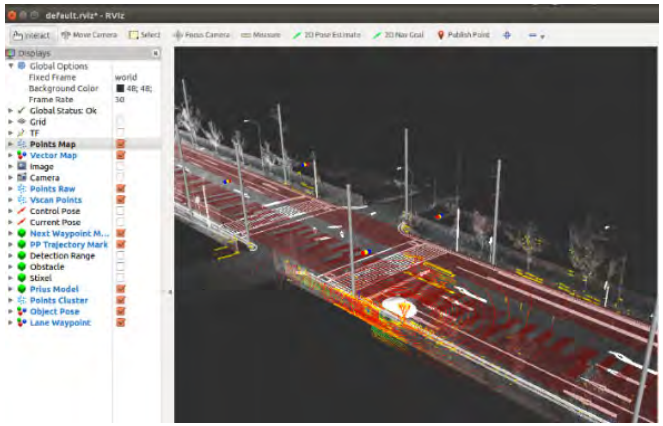
例



分散システム



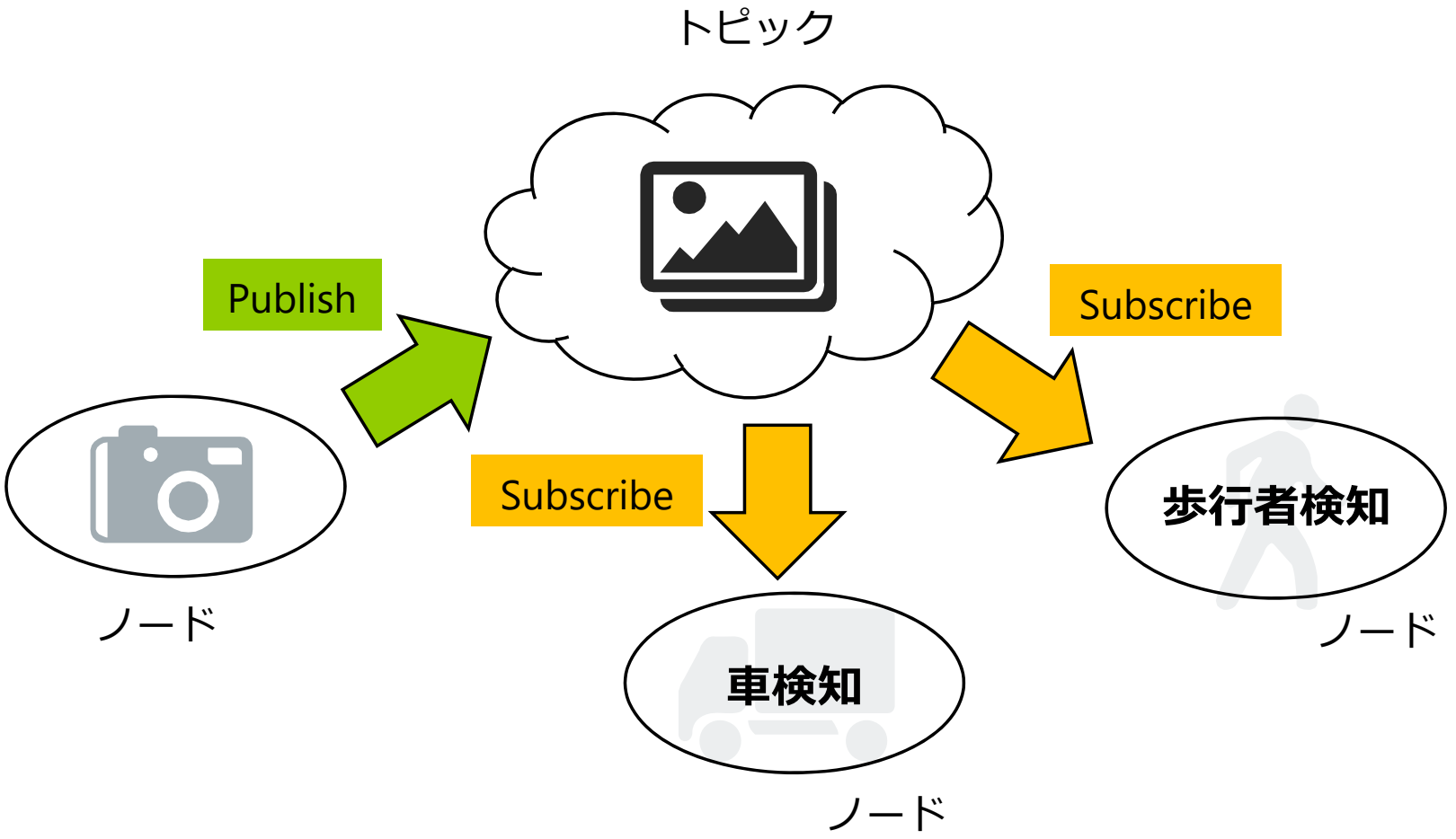
視覚化・シミュレーション



Publish / Subscribe モデル

処理を**ノード**として分割・管理し、**トピック**を介してデータのやり取りを行う。

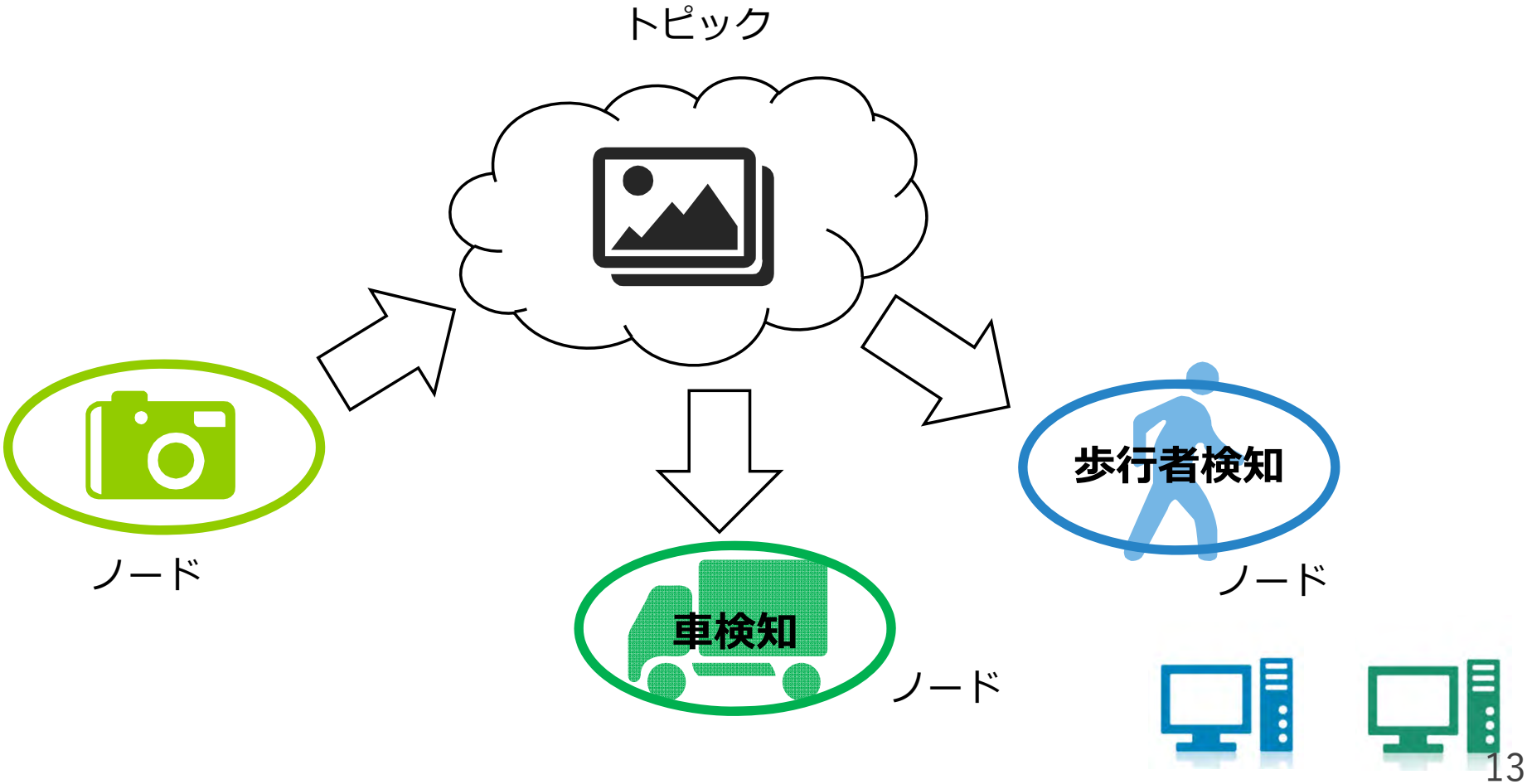
➡ 再利用性・生産性の向上, 分散環境への高い親和性, 障害分離



Publish / Subscribe モデル

処理を**ノード**として分割・管理し、**トピック**を介してデータのやり取りを行う。

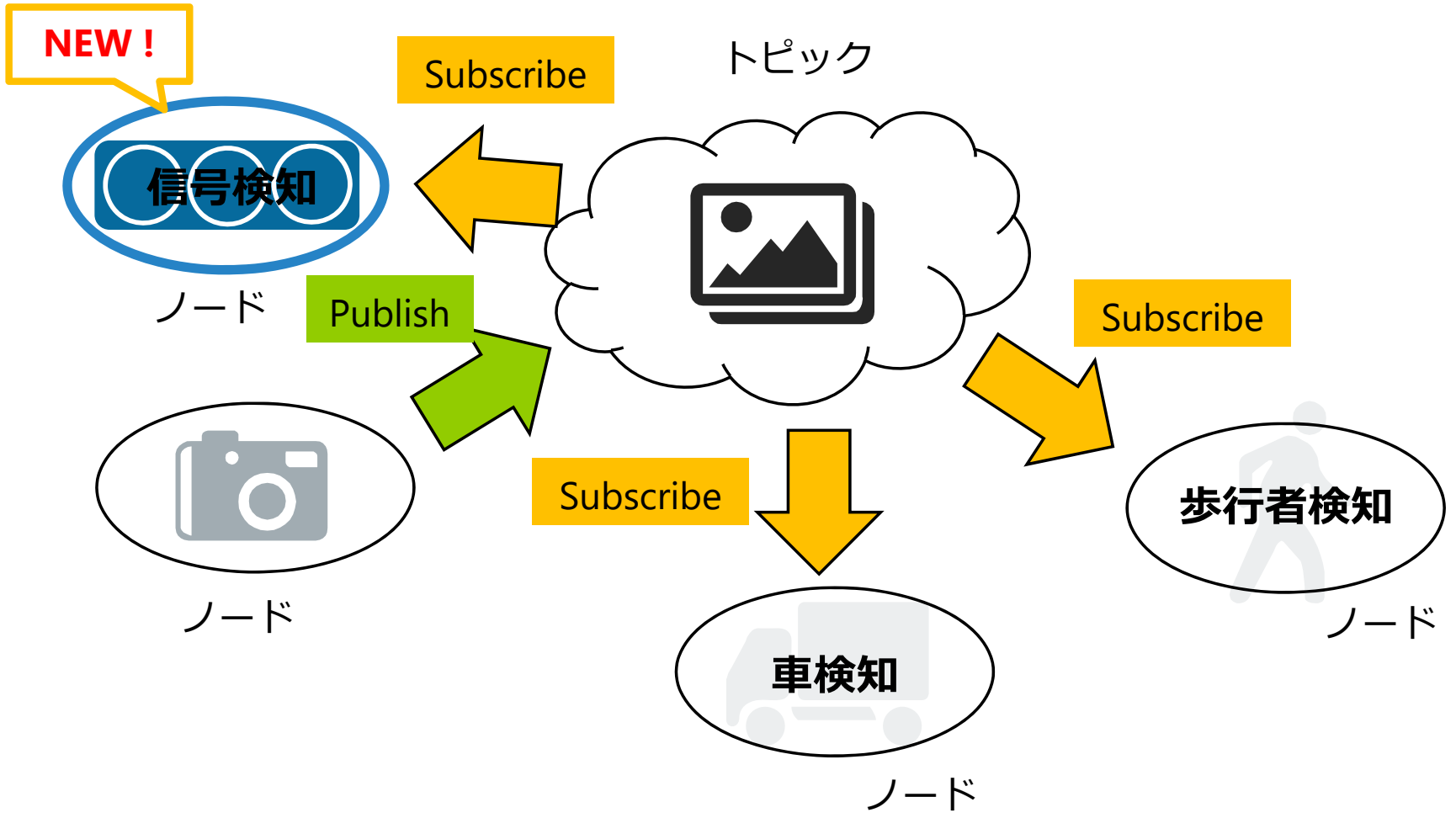
➡ 再利用性・生産性の向上, 分散環境への高い親和性, 障害分離



Publish / Subscribe モデル

処理を**ノード**として分割・管理し、**トピック**を介してデータのやり取りを行う。

➡ 再利用性・生産性の向上, 分散環境への高い親和性, 障害分離



ROS の 特長

データの保存：rosbag

実データ（トピック情報）を保存可能

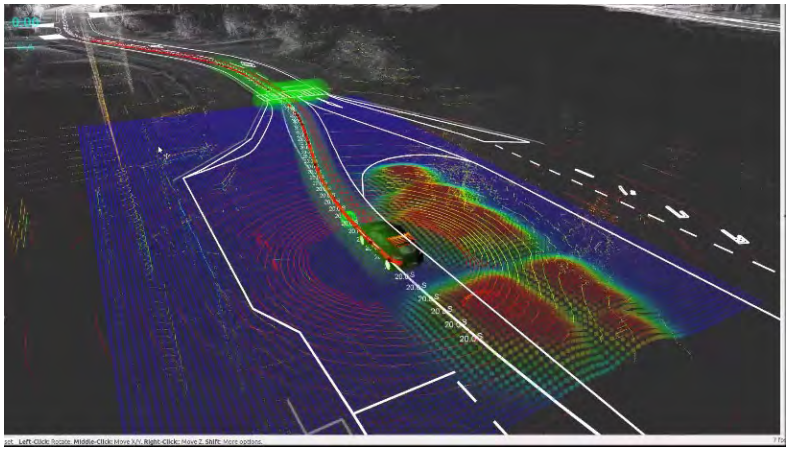
視覚化・シミュレーション

RViz：3D視覚化ツール

簡単にシステム状態を視覚化可能

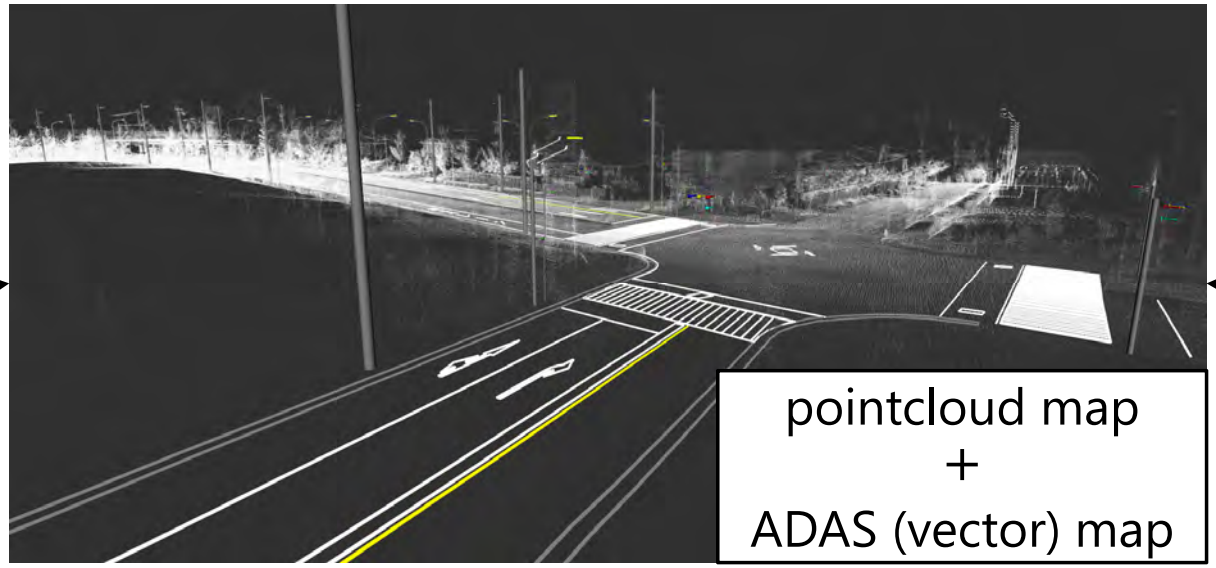
［再生データ］

- 記録したセンサデータ（**rosbag** ファイル）
- 指定した値のデータ



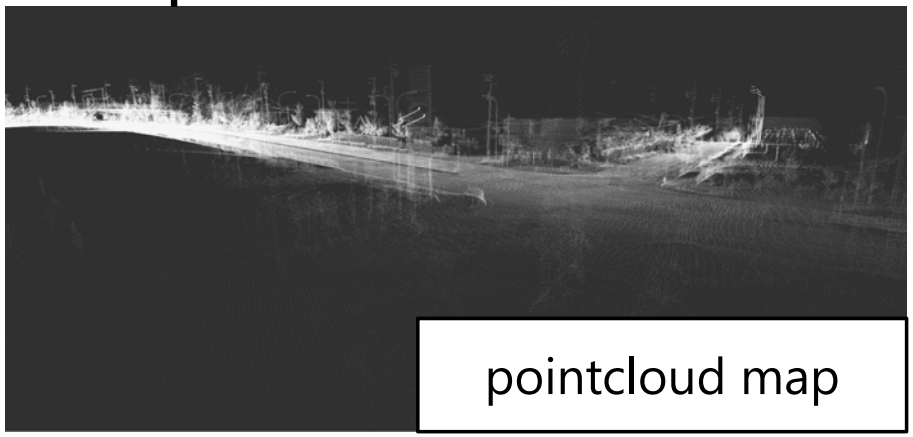
AutoWare

高精度３次元地図



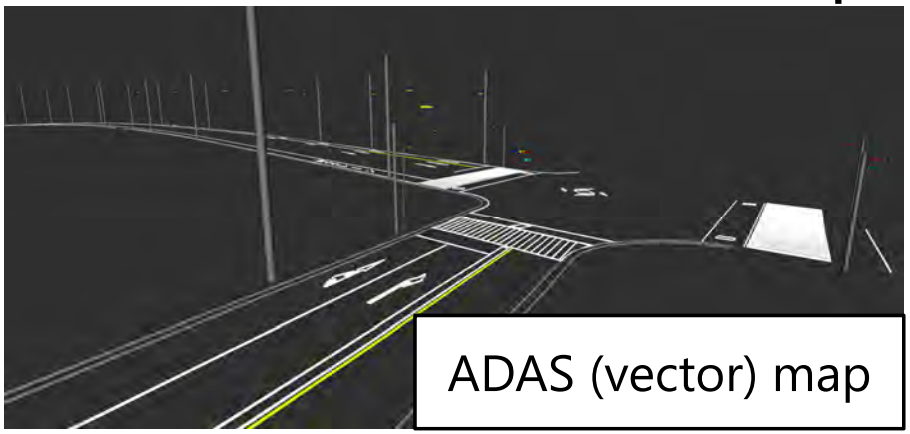
pointcloud map
+
ADAS (vector) map

●ポイントクラウド地図
- ✓ 3次元座標(緯度・経度・標高)
- ✓ RGB値



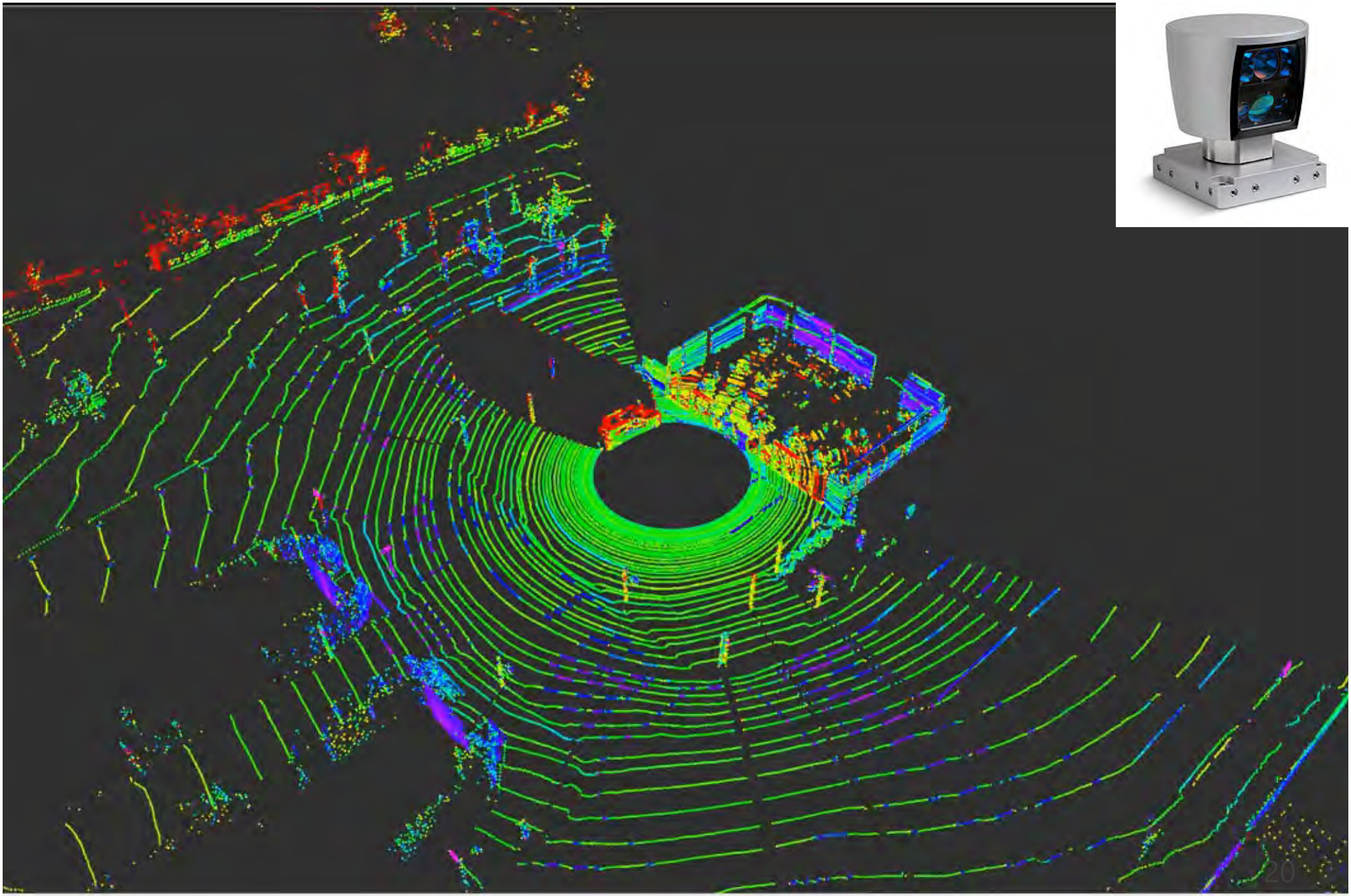
pointcloud map

●ADAS地図 - 点群地図から地物を抽出
- ✓ 信号、路面標示 etc.



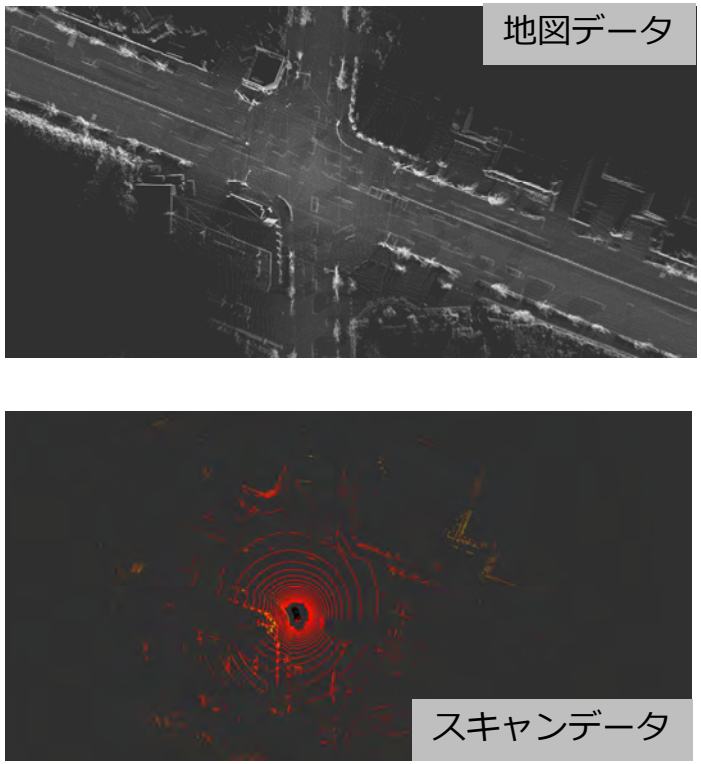
ADAS (vector) map

センシング：LiDAR



LiDARによる自己位置推定

地図データとスキャンデータがきれいに重なる座標変換を計算し、
地図内の位置・向きを算出

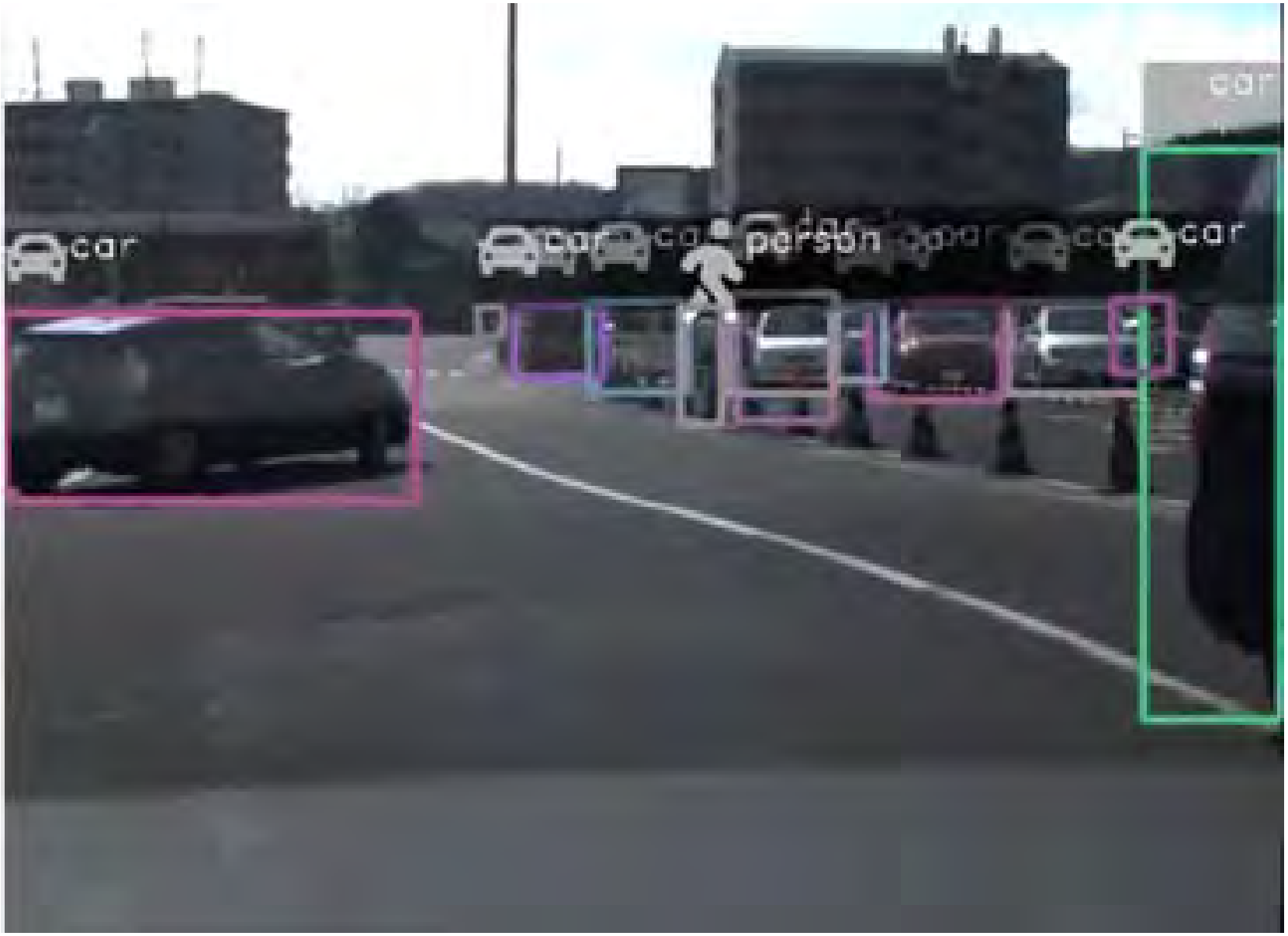


3次元地図とスキャンデータの座標変換を計算
車両の位置・向き

代表的なスキャンマッチングのアルゴリズム

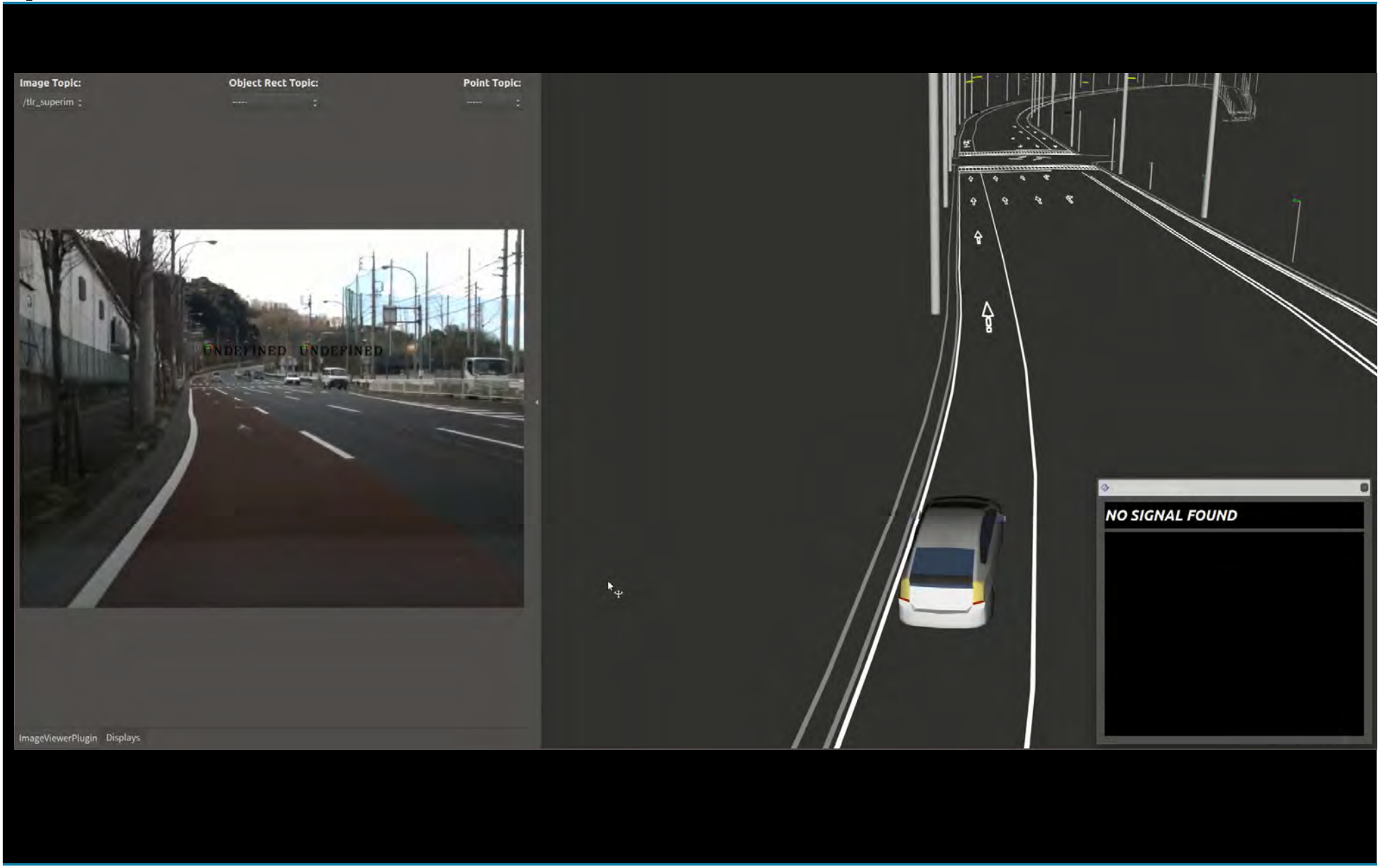
- ICP (Iterative Closest Point) - P.J. Besl et al. (1992)
- 2D-NDT (Normal Distributions Transform) - P. Biber et al. (2003)
- 3D-NDT - E. Takeuchi et al. (2006) , M. Magnusson et al. (2007)

ディープラーニング（YOLOv3）を用いた物体認識：最新



https://www.youtube.com/watch?v=KFfD3Mkkz4Y

Autowareの信号認識

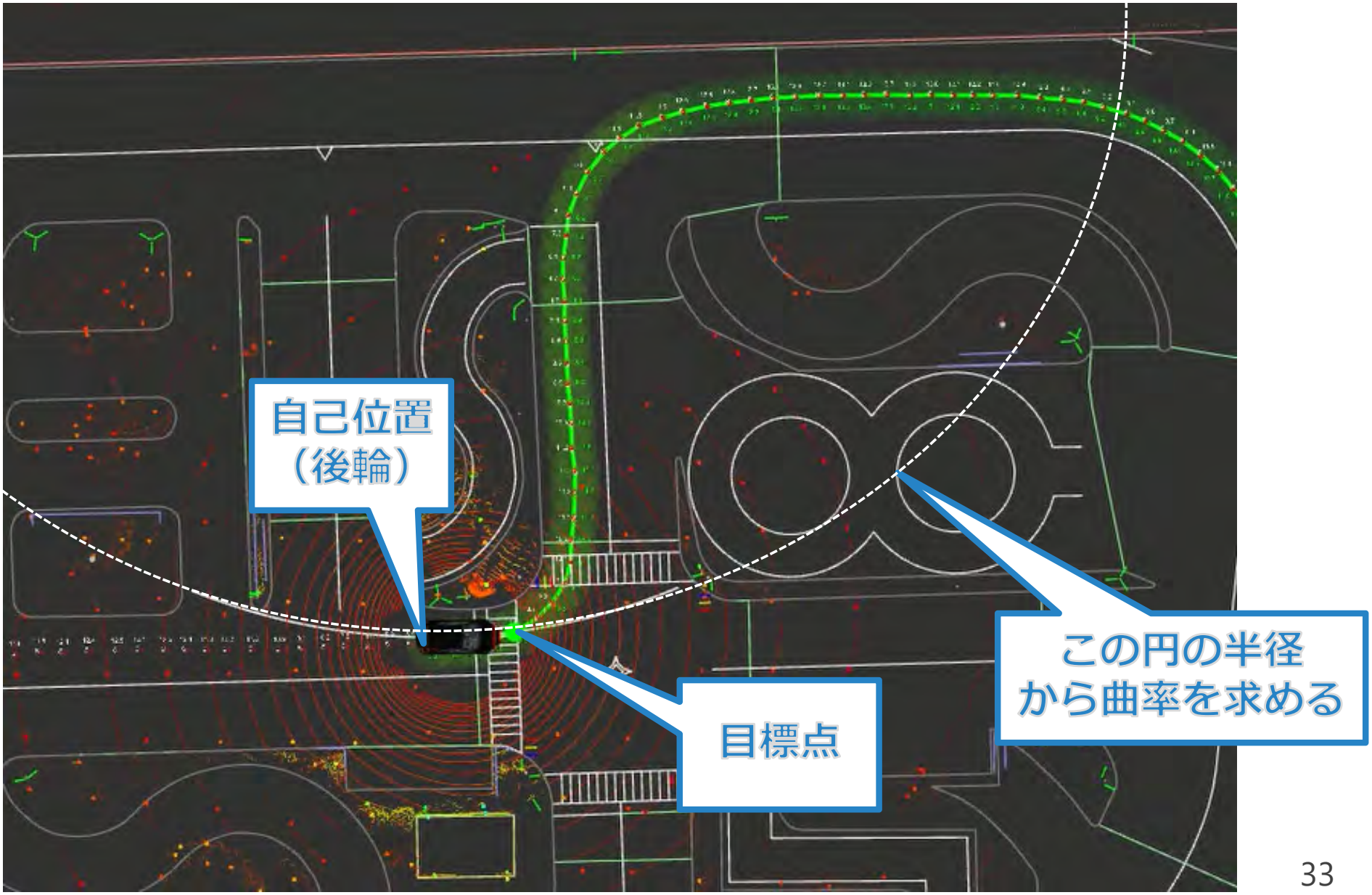


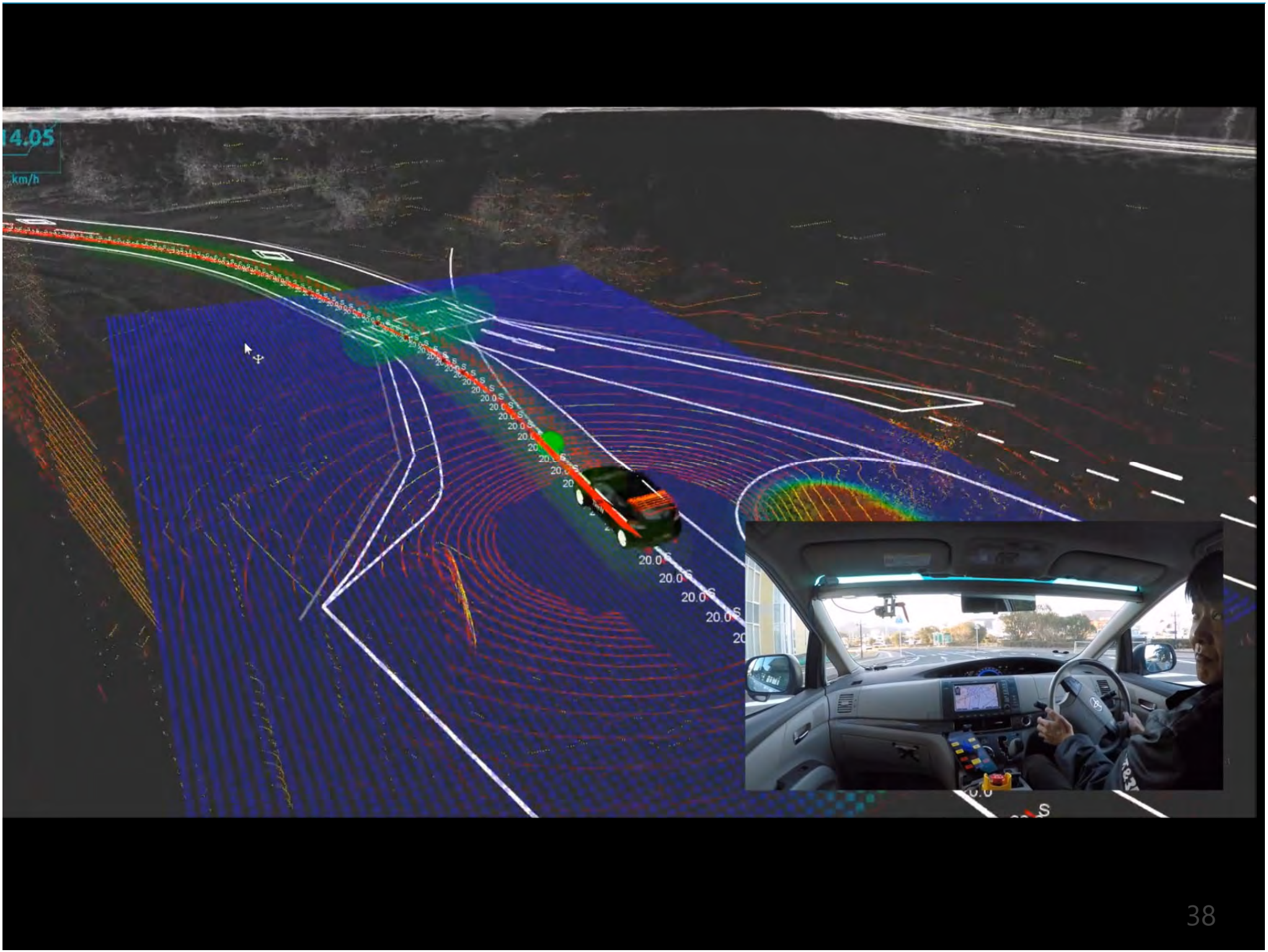
https://www.youtube.com/watch?v=KmOdBms9r2w

Pure Pursuit

※ R Craig Coulter. "Implementation of the Pure Pursuit Path Tracking Algorithm". Technical Report CMU-RI-TR-92-01, Robotics Institute, Pittsburgh, PA, January 1992.

経路上の目標点と自己位置の情報を基に車両制御信号を計算




14.05
km/h

自動運転ソフトウェア:
ROS 2への移行理由

Geoffrey Biggs

Tier IV

ROSとは

- ロボット用のソフトウェアプラットフォーム
- ロボット分野で大成功
  - 様々なロボットで利用されている
  - 研究分野で人気である
  - ベンチャー会社でも大手メーカーでも利用されている
  - 購入可能なROSに基づいて開発された製品もある

ROSはどこからきたか

- Willow Garage
  - Scott Hassanの創業したロボティクスベンチャー
- Willow GarageのPersonal Robotics Program
  - ロボット研究を支えるために
  - 共通なハードウェアプラットフォーム
  - 共通なソフトウェアプラットフォーム
- オープンソースが中心

ROSはどこからきたか

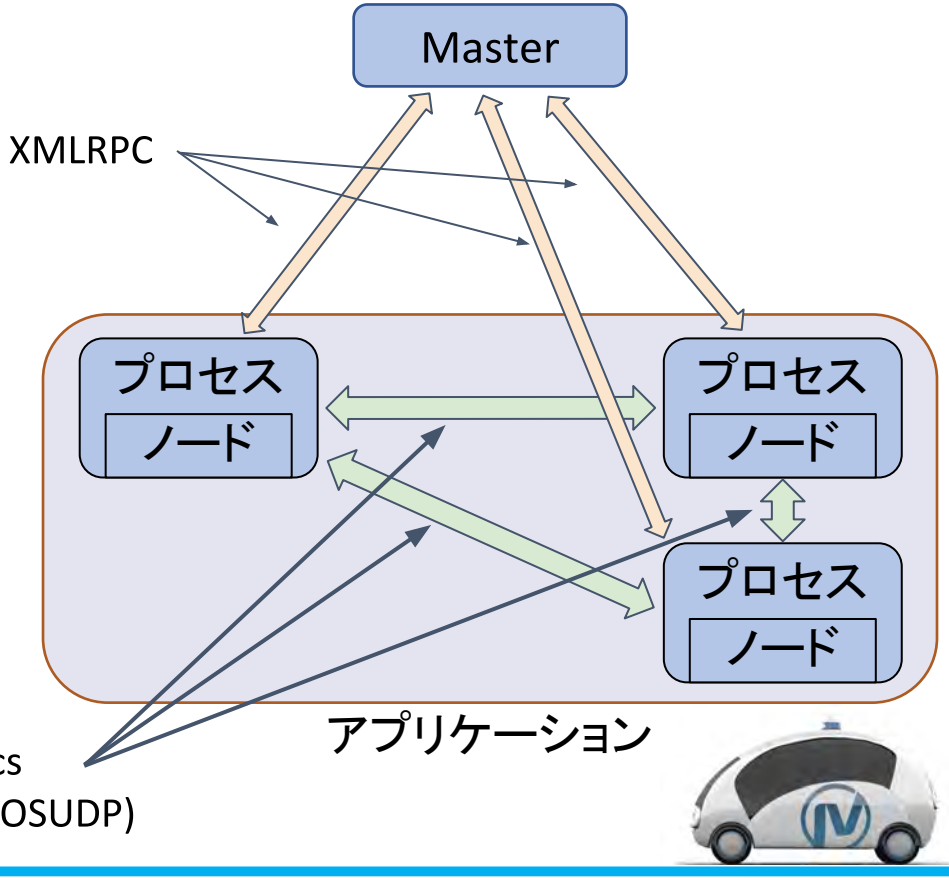
- とあるユースケースのために開発
  - 独立なロボット
  - 最先端なIntelCPU搭載
  - 大量のメモリー搭載
  - リアルタイム制御はハードウェア側
  - 研究用

ROSはどこからきたか

- 現在はOpen Source Robotics Foundationが中心
- Willow Garageからのspin out
- 数十人のソフトウェアエンジニア等
- NPO

Terminology

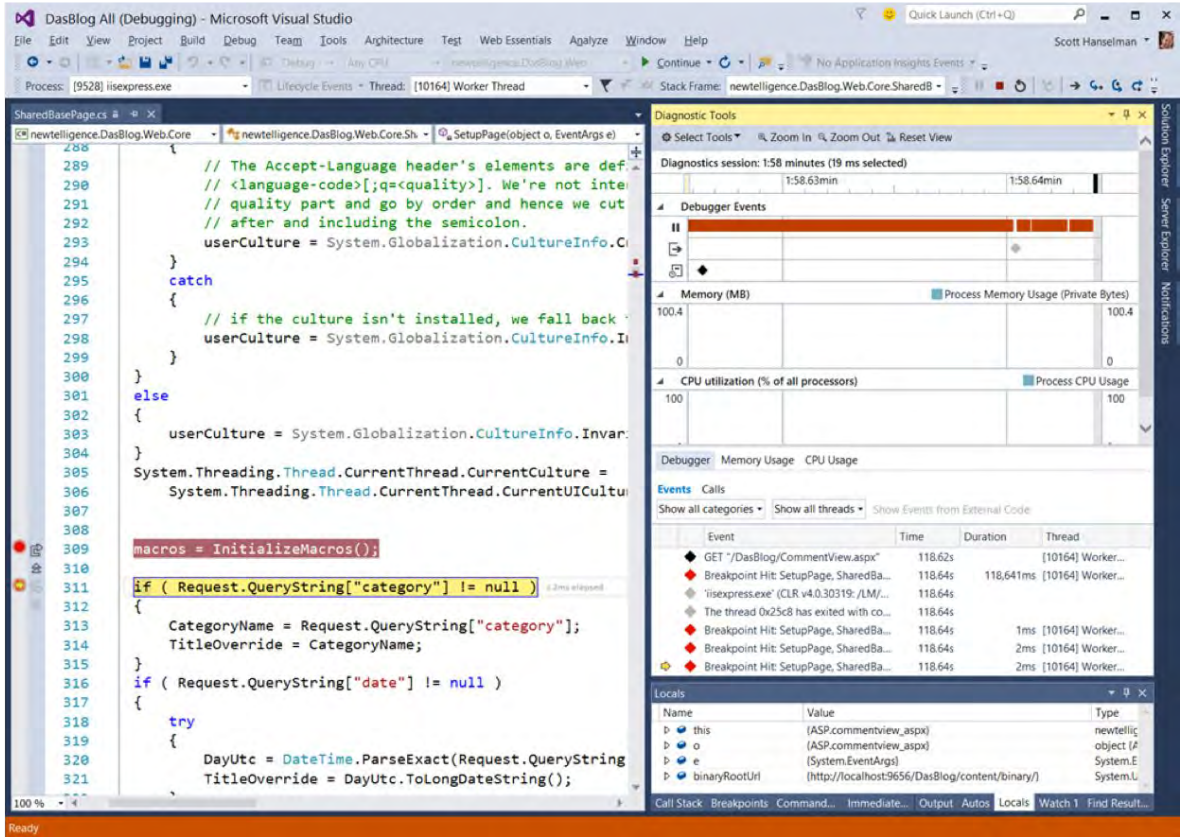
- Master（マスター）
- Node（ノード）
- 通信プロトコル
  - ROSTCP、ROSUDP
  - XMLRPC
- Topic（トピック）



ROSの特徴

- ミドルウェアだけでなく、オープンプラットホーム
  - 他社の手を無料で借りる
  - 独自で開発すると他が届かない機能がもらえる
- 開発をサポートする機能
  - Introspection機能
  - デバッグツール
  - ソース再利用サポート
  - 共通データ型
  - …
- ツールは豊か
  - ロボット開発にいいツールは必須
  - ツールの例：デバッグツール

デバッグ：PC



Tier IV
Intelligent Vehicle

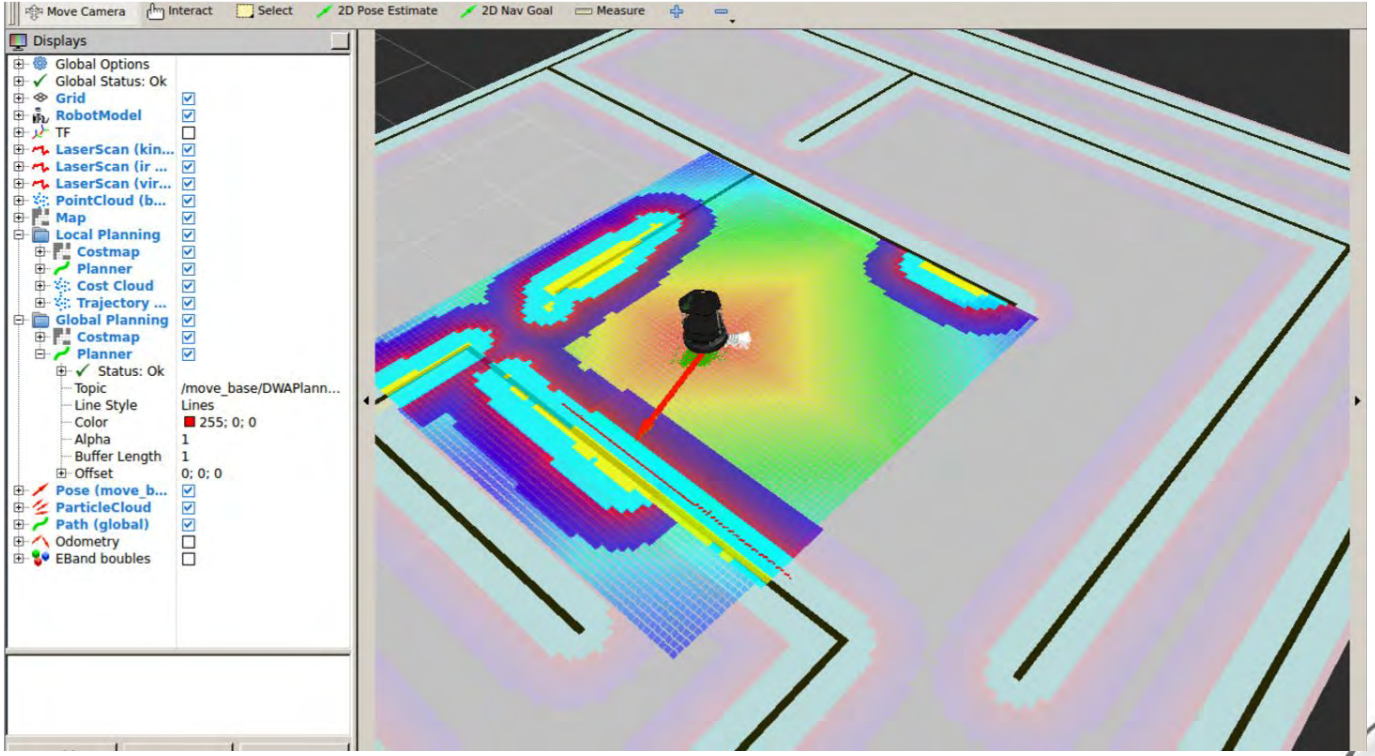
デバッグ：ロボット

0.47047596160651484, 0.012463983269873857, 0.13185027912023128, 0.5380365386000314, 0.403864612045705,
0.06069930337048779, 0.3691795213234019, 0.8515692765843788, 0.07369674423123 84, 0.2843486311730886,
0.8538529785225724, 0.3129984654314725, 0.3584207454827225, 0.45378466374113 78, 0.7561766125048414,
0.8508870244042188, 0.0520696074562399, 0.8765451038148542, 0.15299405008975697, 0.5867355890849806,
0.9669883187750915, 0.41628598565655384, 0.34816285093525845, 0.9728408650406579, 0.41829494089973274,
0.579997374892018, 0.5531830445281459, 0.24530608285472844, 0.38526280587020356, 0.2210817119943037,
0.07975484090387797, 0.6433789010590776, 0.14258622222962336, 0.47068371292023214, 0.021146762861280366,
0.7809218654795851, 0.059743545404254084, 0.7174453572838055, 0.11796690651681152, 0.45170824264501075,
0.3463075513634315, 0.12329222545582419, 0.9238715097652316, 0.20246585186042165, 0.07287853589316085,
0.8857115222065027, 0.2607754782454512, 0.7871998842710477, 0.3123418728541889, 0.24793655686384475,
0.03936840867259228, 0.15830275932784887, 0.0588537758302089, 0.6905061474343309, 0.45187144905773735,
0.3137586806320821, 0.3928776902569401, 0.18194845109045765, 0.7772116865207355, 0.10878199902185848,
0.6046192473442614, 0.3783574047058046, 0.9335545367770111, 0.5931631033298297, 0.8227864249862641,
0.37984882478112314, 0.35863612685228197, 0.712004279109606, 0.3414544916392299, 0.4118358463966818,
0.7092836179969 54, 0.14020958524673632, 0.03867697529807812, 0.5240273928062406, 0.631862692442692,
0.9698634206694162, 0.6034612197461879, 0.3218208000413312, 0.6832981693054933, 0.6050553280734954,
0.6604449894889293, 0.3442873321468821, 0.18196942376936986, 0.4199897381622488, 0.8804476148633672,
0.1922718532155836, 0.963177730081603, 0.21728977473403277, 0.20754416331944092, 0.30192492471822563,
0.190634027338090 54, 0.9498797020928332, 0.6673141005665001, 0.45745969941649967, 0.4351040209987119,
0.5928193275952122, 0.6823498640502099, 0.7442921879931558, 0.00932650033833149, 0.11983479696483657,
0.0519672566623508, 0.36971891569137594, 0.190717653457389, 0.74929028390 65593, 0.5602249232567452,
0.6585780171293569, 0.08658888622314564, 0.6945120787132953, 0.55806657002 78967, 0.48146812759625746,
0.4540375145339196, 0.8059387960368088, 0.8555468352647819, 0.57371104345030 95, 0.35666621246462693,
0.06659849272034857, 0.21051537392984276, 0.731280246174109, 0.91513652688725 26, 0.37390717986528854,
0.32541148512546336, 0.5594068298257496, 0.08044184535792642, 0.55912073047 51257, 0.20915450867051832,
0.02534903593989679, 0.19416932780283314, 0.34906251623185813, 0.84321528577 63088, 0.8664024388302471,
0.2840496778118137, 0.2781658794647851, 0.2465128654531562, 0.9988375836561965, 0.887499520253597,
0.027578461404808352, 0.29371750902319593, 0.8873891214375212, 0.914573150995103, 0.000652243172693745,
0.06772123956909537, 0.9597290829459296, 0.7714181981544204, 0.32507290632274355, 0.11058804894490903,
0.99981521387735 76, 0.6229449431503 89, 0.3816913626678231, 0.7083424647825276, 0.114043281363730 76,
0.21818485912970653, 0.08168144717155512, 0.6732894753601069, 0.5991484405675498, 0.026001431290659682,
0.7854449857322512, 0.2454220869235253, 0.9264245146883056, 0.9350749518287534, 0.9652777708676377,
0.8655942624669641, 0.27395894599419734, 0.0785308404808881, 0.3250820262765106, 0.5039220523760805,
0.4985074812618765, 0.01020455824663058, 0.5881052368037296, 0.01781120958533622, 0.27810744561068323,
0.4818559083236109, 0.53535289994413 7, 0.6768776195049253, 0.16544968523832215, 0.5330947114782651,

デバッグ

- ロボット開発で特に難しいところ
  - データが大量
  - 世界は停止できない
- 世界をロボットのように見えると問題の原因が明確に
- ライブまたは再生で見たい
- 1フレームづつで見ると終わらない

デバッグ：データの可視化



ROSの弱点

単一障害点

起動順番がランダム

品質ってなに？

Ubuntuバンザイ！

独立派ノード

リアルタイムなんてない

スーパーコンピュータが好き

安全性？認証？

100%自家製

それはなんでしょうか？

壊れやすい通信

ROSの弱点

単一障害点

起動順番がランダム

品質ってなに？

Ubuntuバンザイ！

すべて自動運転システムの敵

独立派ソフトウェアエンジニアなんてない

スーパーコンピュータが好き

安全性？認証？

100%自家製

それはなんでしょうか？

壊れやすい通信

ROS 1はだめならどうすればいい？

ROS 2に移行する！

（色々がよくなっているから）

ROS 1: サポートされるOSは少ない

- POSIX型OSだけ
- Ubuntuではないと使いにくい
- 組み込みOSでの利用は<u>不可能</u>

Ubuntu in your car!

ROS 2: 様々なOSがサポートされる

- サポート級制度
  - Tier 1: OSRFがサポートする
  - Tier 2: 別の会社がサポートする
  - Tier 3: 動くはず
- 組込みOSなどのサポートが準備中
  - eSOL eMCOS
  - VxWorks
  - NuttX
  - …

Dashingのサポートプラットフォーム

Tier 1:

Ubuntu (amd64、arm64)

Windows (amd64)

Mac OS (amd64)

Tier 2:

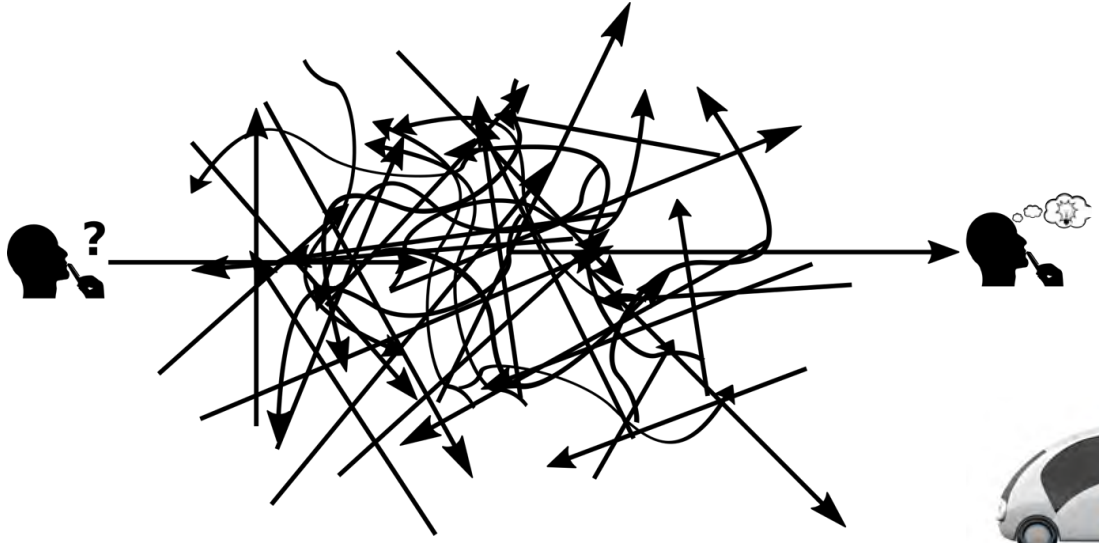
Ubuntu (arm32)

Tier 3:

Debian (amd64, arm64, arm32)

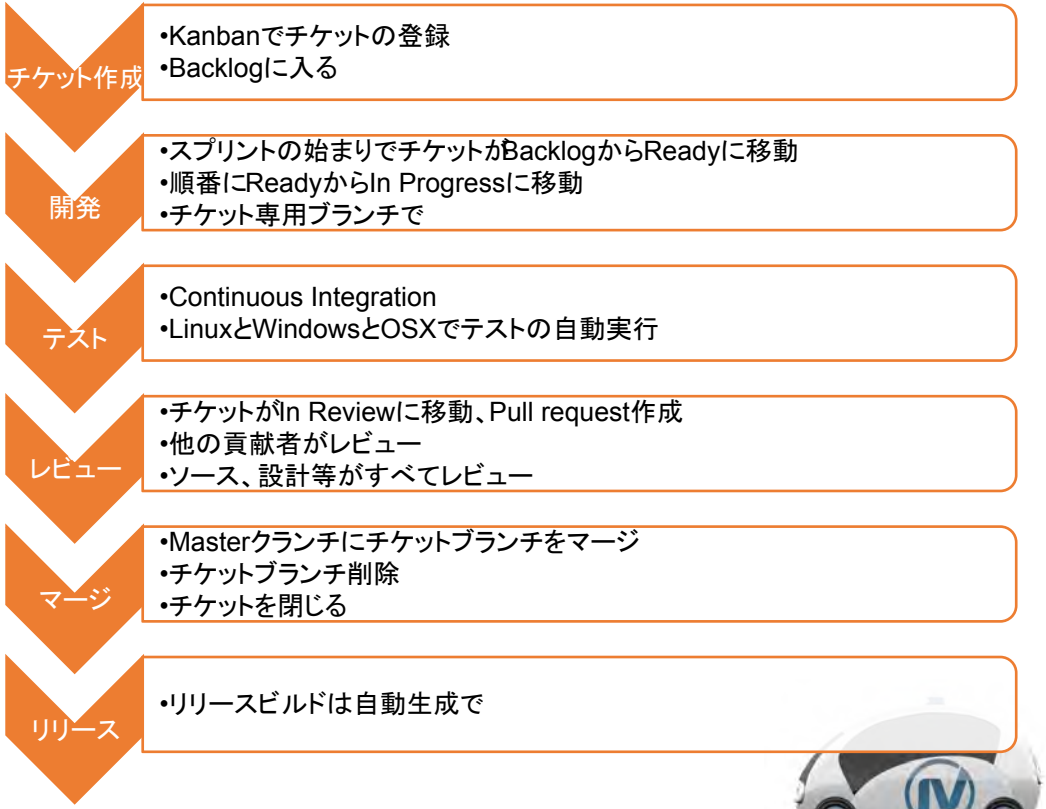
ROS 1: QAはあとでいい

- 200X時代のオープンソースソフトウェア
- 開発プロセスはほとんど定義なし
- CIは基本のみ

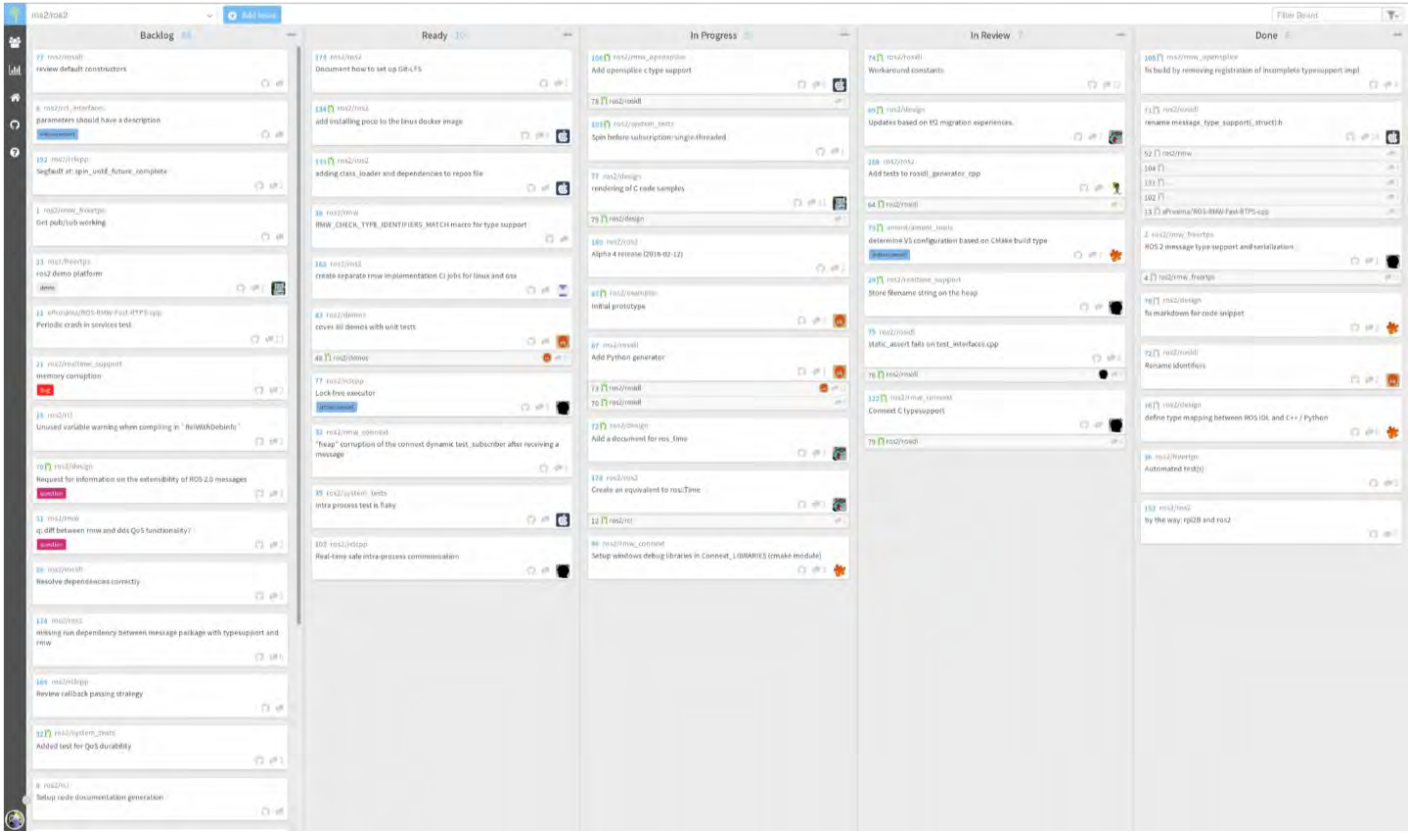


ROS 2: QA中心

- 開発プロセスの定義
  - アジャイル
  - TDD
- Kanbanで管理
- CIがマスト
  - サポートプラットフォームは全部あり
- 設計議論がオープン

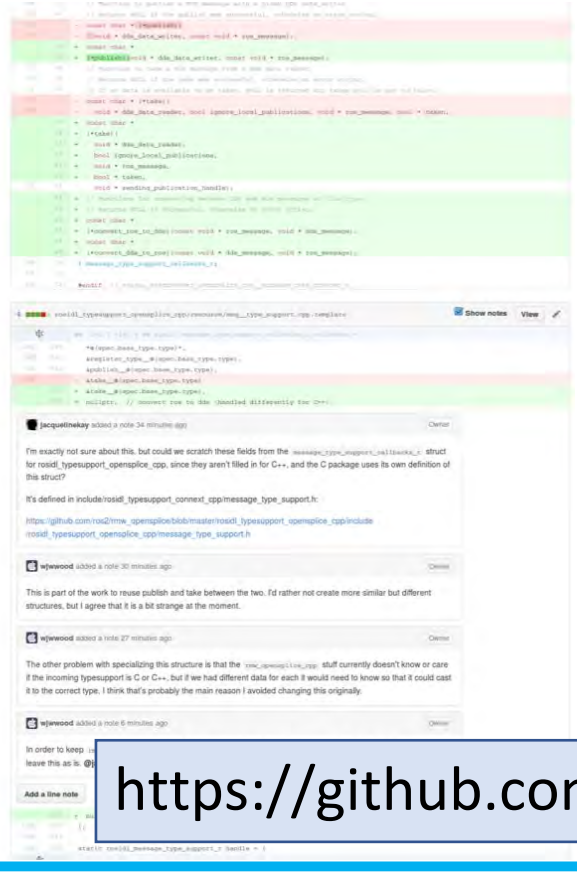


ROS 2: QA中心



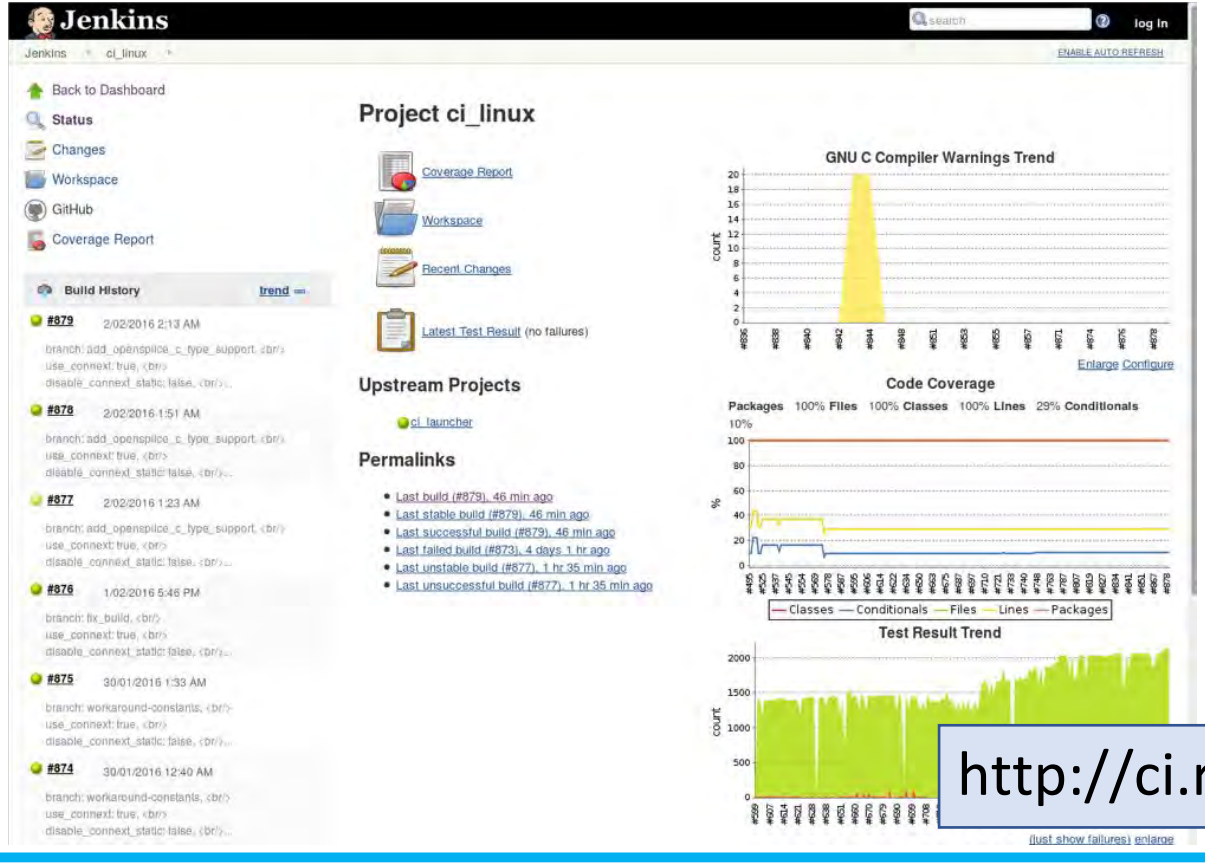
Tier IV
Intelligent Vehicle

ROS 2: QA中心



https://github.com/ros2

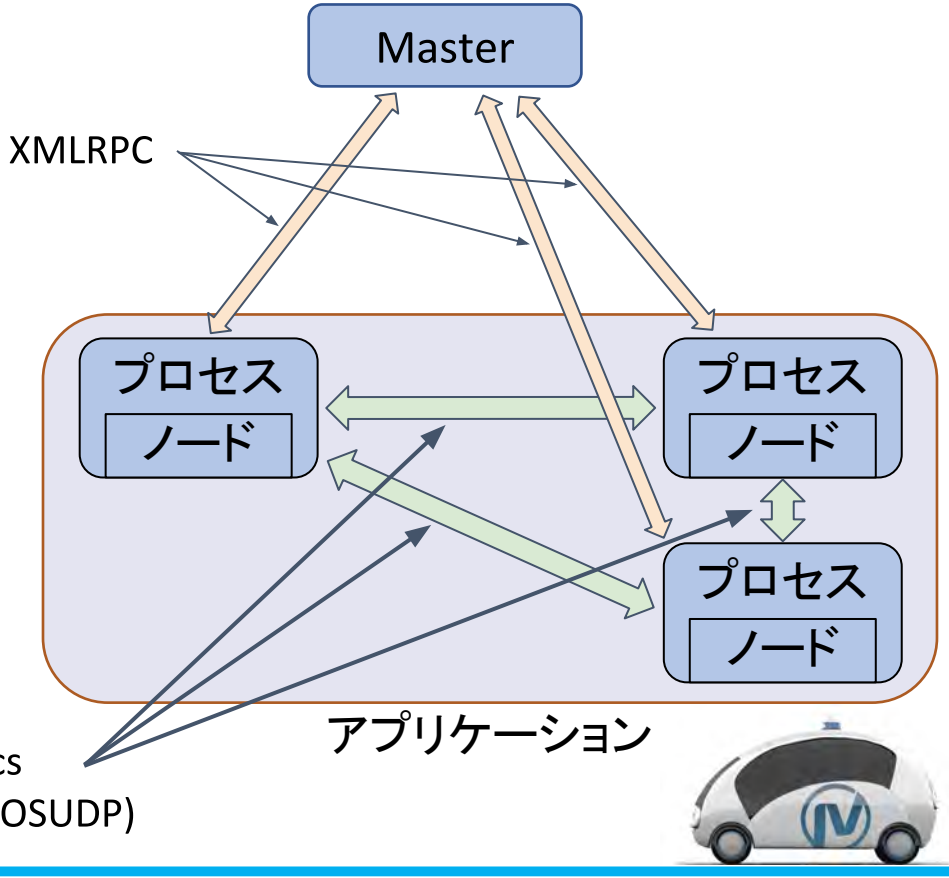
ROS 2: QA中心



http://ci.ros2.org/

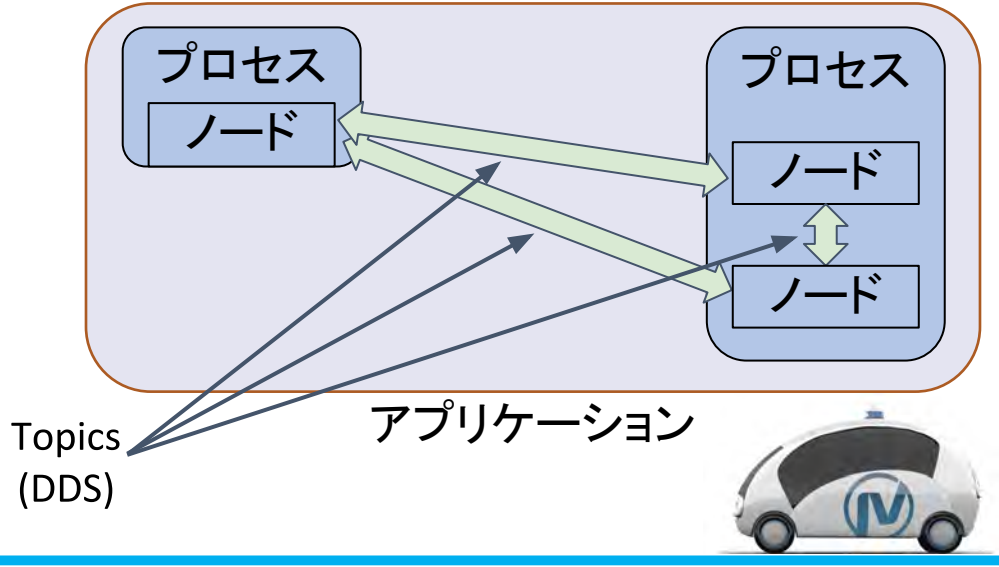
ROS 1: 自家製通信プロトコル

- Masterが必須
  - パラメータサーバ等も
- メタデータ通信はXMLRPC
- データ通信は自家製プロトコル「ROSTCP」
- ノードは別々のプロセスで実行

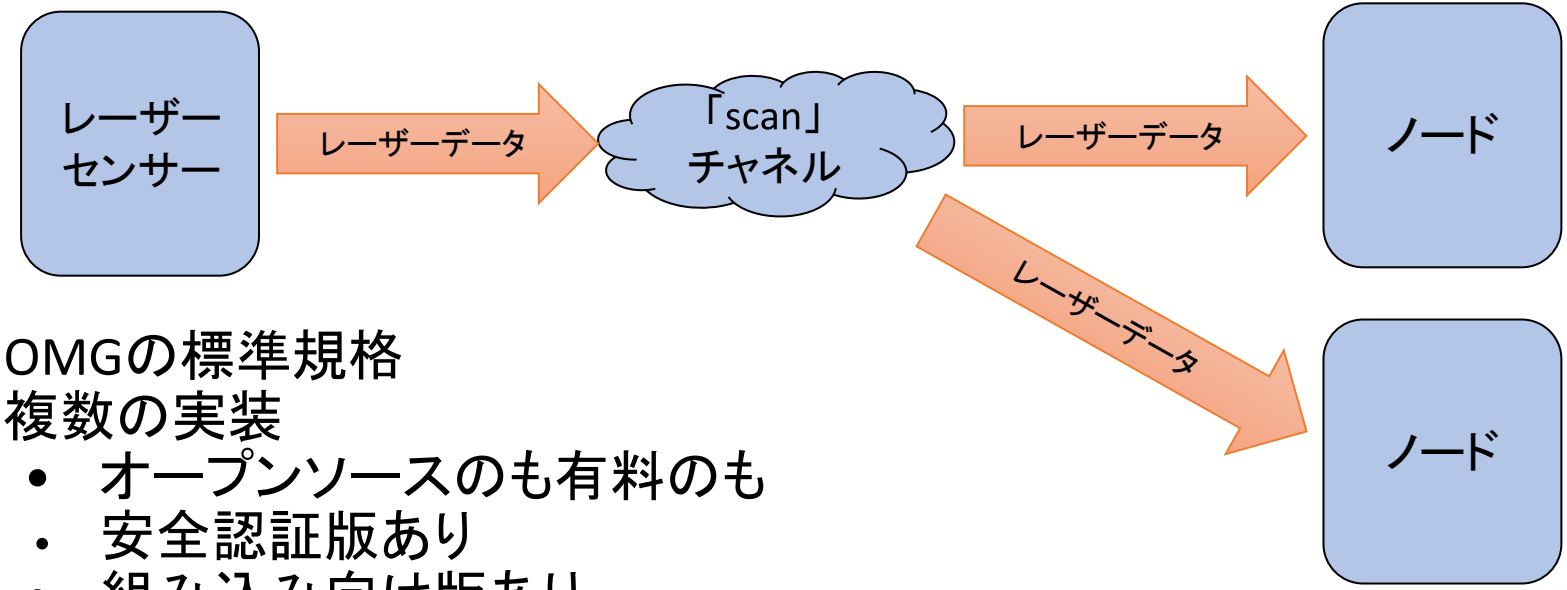


ROS 2: 標準化されたプロトコルの導入

- ROSTCP、ROSUDP、XMLRPCを排除
- Data Distribution Serviceを導入
  - メタデータ
  - アプリケーションデータ

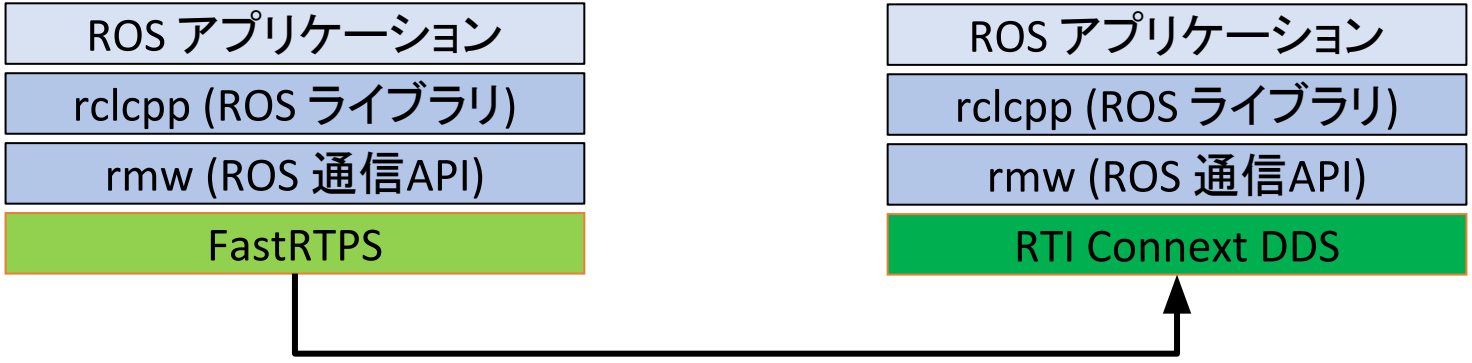


DDS（Data Distribution Service）



- OMGの標準規格
- 複数の実装
  - オープンソースのも有料のも
  - 安全認証版あり
  - 組み込み向け版あり
  - 超分散向けあり

DDS実装間の互換性



サポートありDDS ベンダー	ライセンス
eProsima FastRTPS	Apache 2
RTI Connext DDS	Commercial
ADLink OpenSplice DDS	Commercial

他のベンダーもある

DDSのQuality of Service設定

ROS 1: チューニングができない

Best-effort
(UDP)

Reliable
(TCP)

ROS 2/DDS: 細かいチューニング可能

Best-effort

Reliable

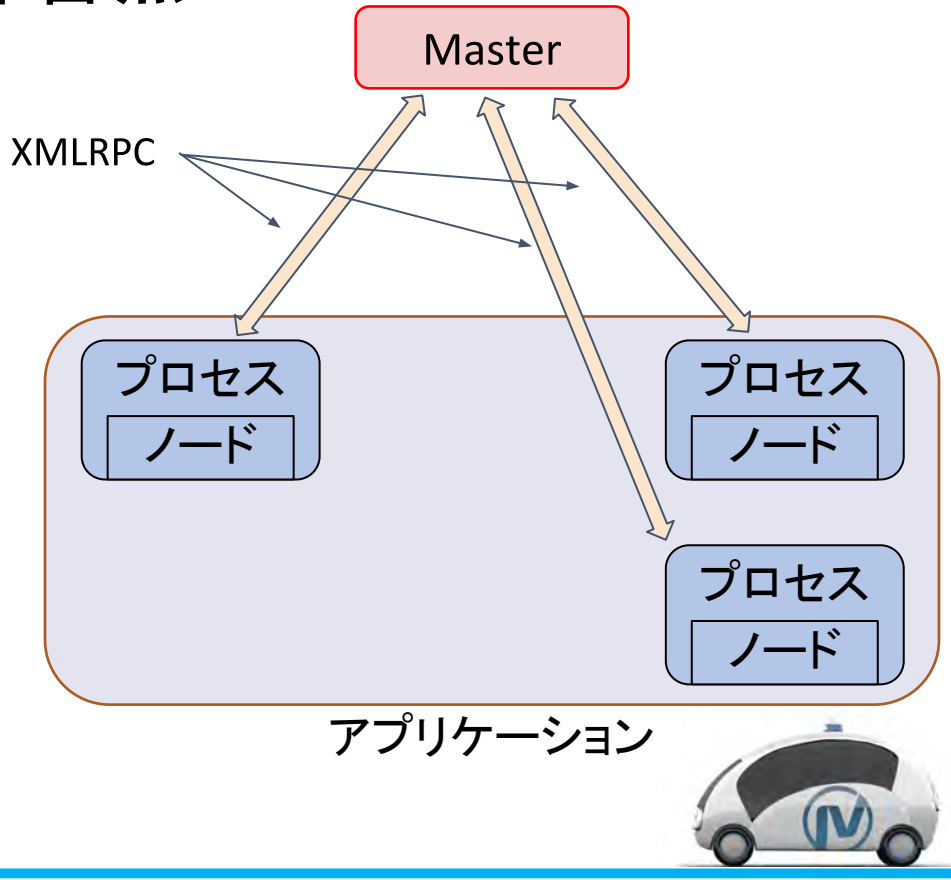
(UDP/TCP/…)

DDSのQuality of Service設定

- 他のQoS設定は様々
  - History
  - Depth
  - Durability
- ユースケースようのディフォルトポリシーセット
  - センサーデータポリシーセット
  - パラメータポリシーセット

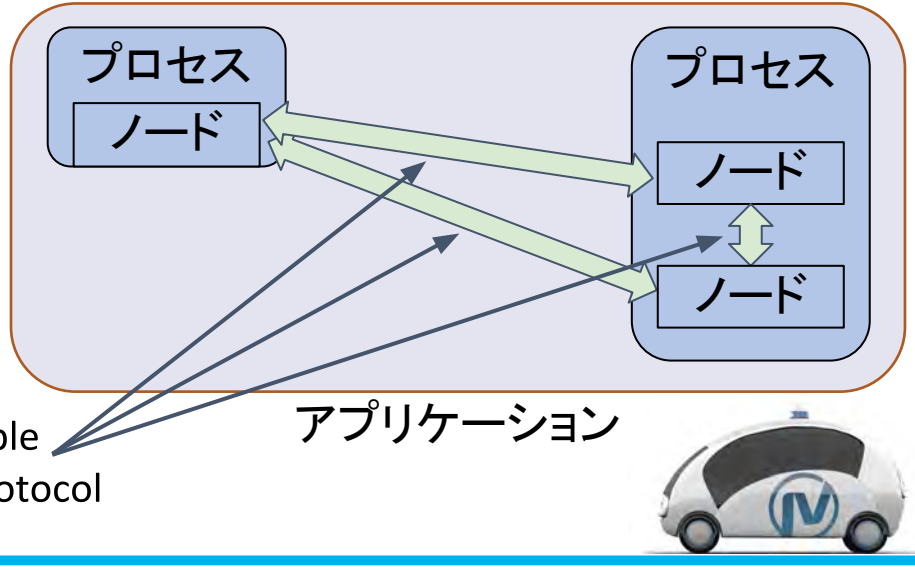
ROS 1: Masterが単一障害点

- Masterが必須
  - ノードの電話帳
  - パラメータサーバ
  - …
- Masterが落ちるとアプリケーションはほぼ終わり
- Masterを再起動したら起動中のデータがない



ROS 2: Masterがない

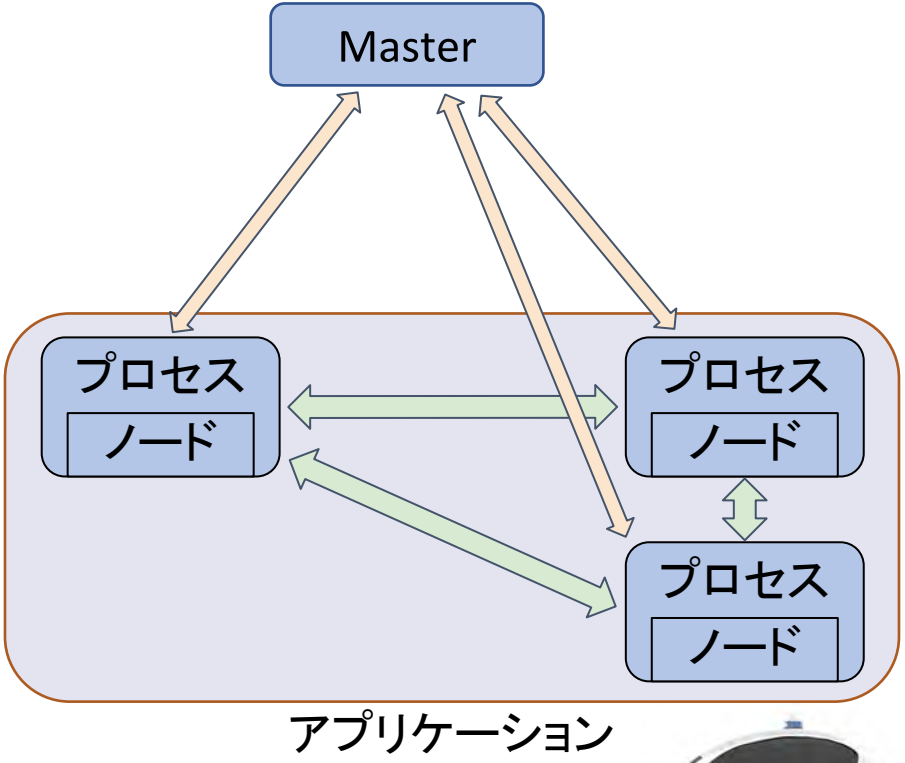
- メタデータはDDSで通信
- 電話帳ではなくてDiscovery
  - 「Simple Discovery Protocol」
- あるノードが途中に
  落ちても復活は可能
- パラメータはノードごとに保存



ROS 1: すべてのノードがプロセス

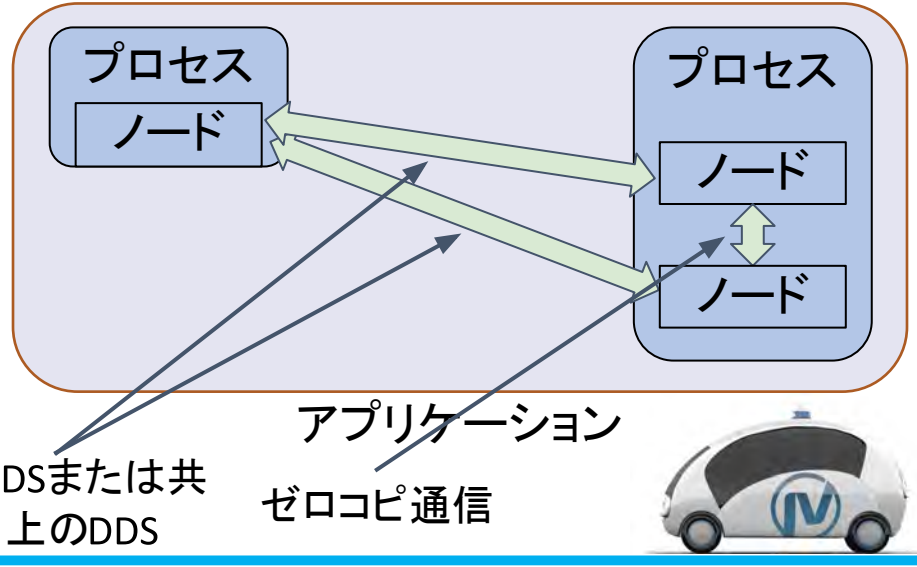
- 一つのノード＝一つのプロセス
- プロセス間でリソースは共有不可能

- 「Nodelet」という仕組みもあるが
  - 別API（実装時間で選択）
  - 壊れやすい
  - 共有ポインター通信のみ
  - QoSなし



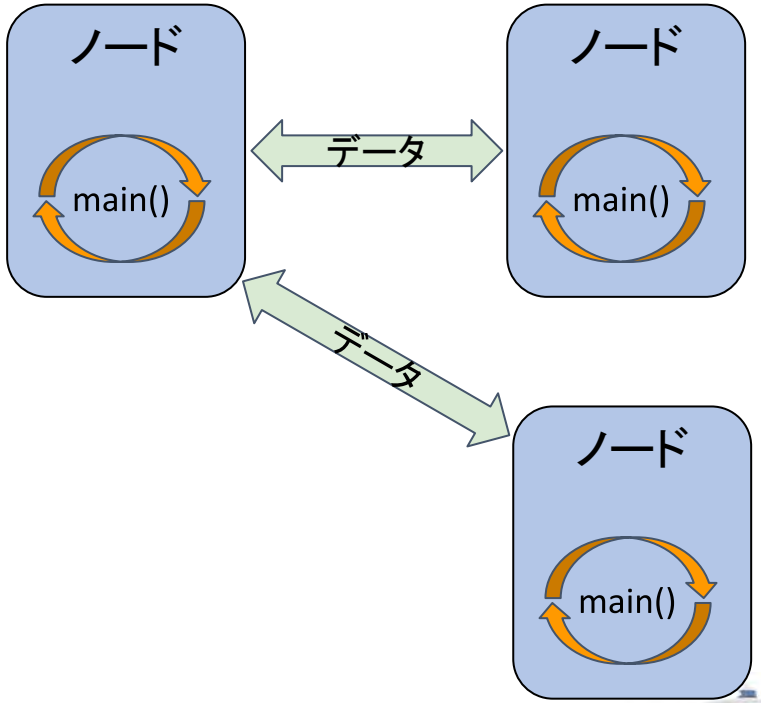
ROS 2: Component Nodes

- ノードはプロセスに統合可能
  - 全部のノードを一つのプロセスに
  - 複数のプロセスを利用（ノード数＞プロセス数）
- コンパイルまたはランタイムで決める可能
- プロセスに統合すると
  - ゼロコピー通信が可能
  - 実行順番が指定可能
  - 平行または連続実行



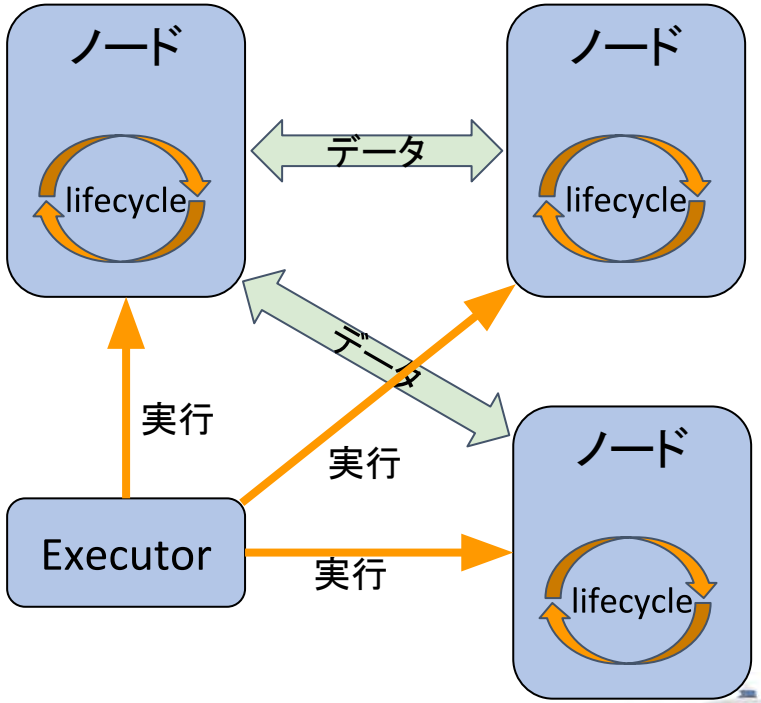
ROS 1: 実行管理不可能

- すべてのノードは
  main関数を持つ
- 起動順番は管理しにくい
- 起動中の実行順番は管理できない
  - 決定的な実行は不可能

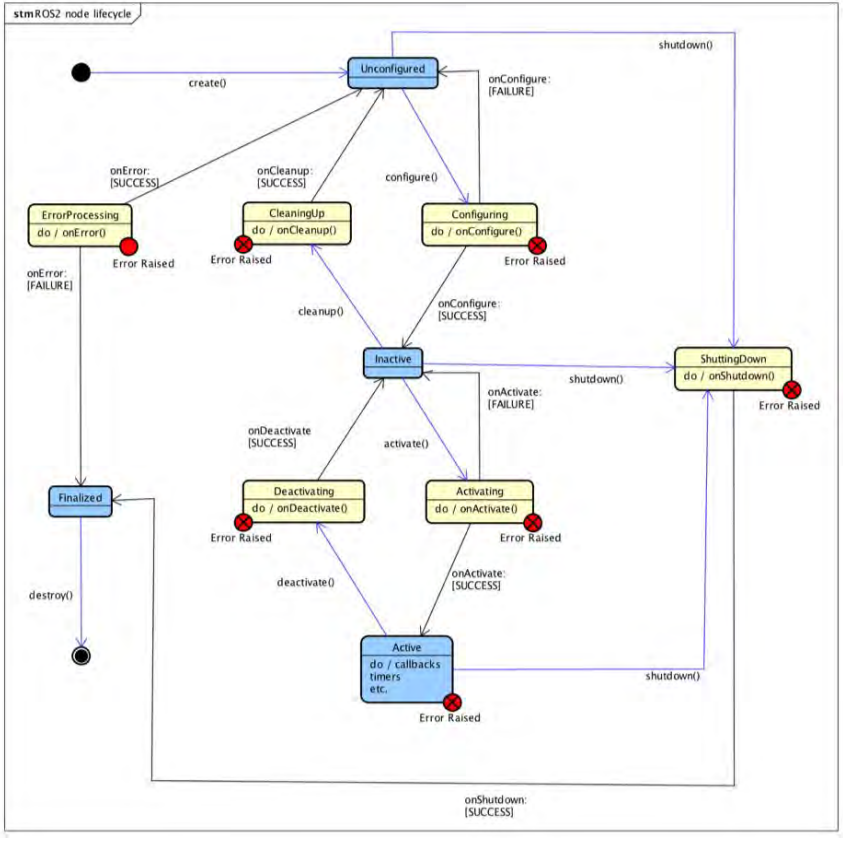


ROS 2: 振舞いと実行の分離

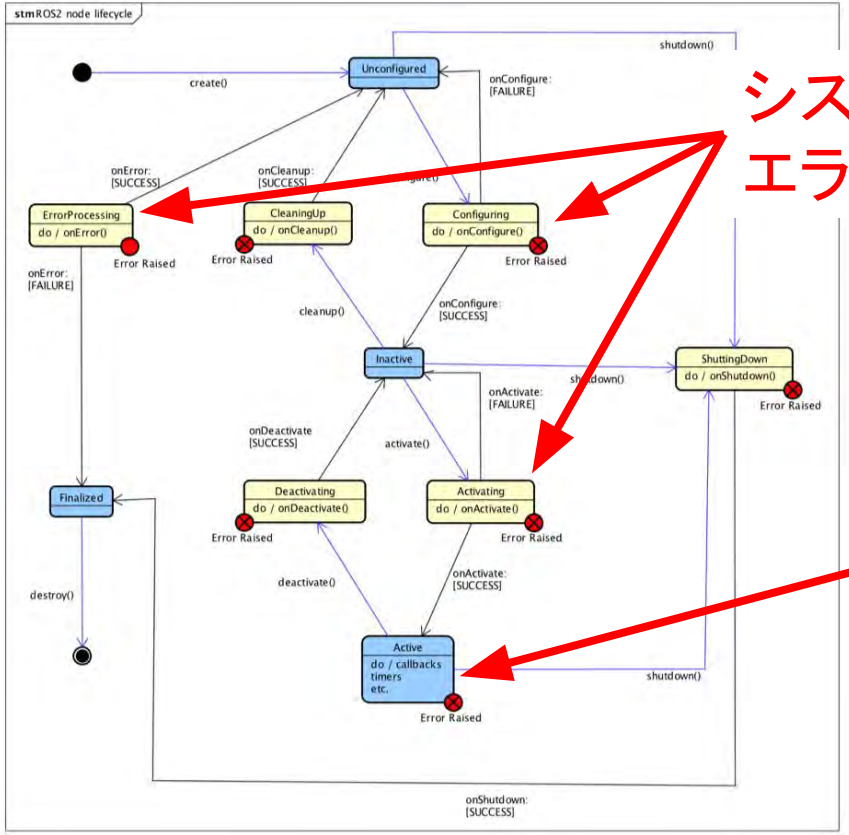
- ノードは振る舞いを定義
- Executorは実行を管理
  - いつ
  - 何を
- ExecutorはROS 2に搭載
  - カスタムも可能
- ノードはLifecycleで実行
  - 起動順番依存しないように
  - エラー管理のために



ROS 2: ノードライフサイクル



ROS 2: ノードライフサイクル



システム管理（セットアップ、エラー状態等）用のステート

このステートのみで本機能を実行する

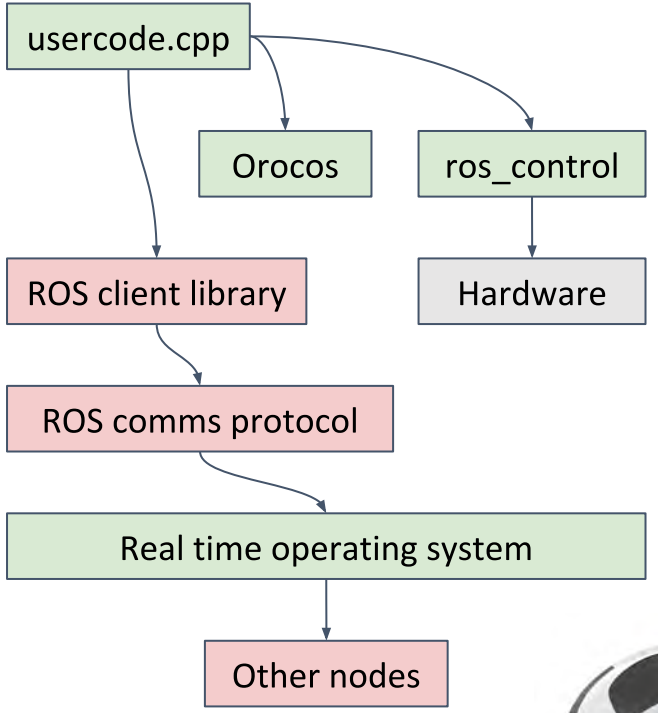
Tier IV
Intelligent Vehicle

ROS 2: ノードライフサイクル

- もしろん、メーン関数でノードの実装はまだ可能!
  - （プロトタイピングに必須だから）

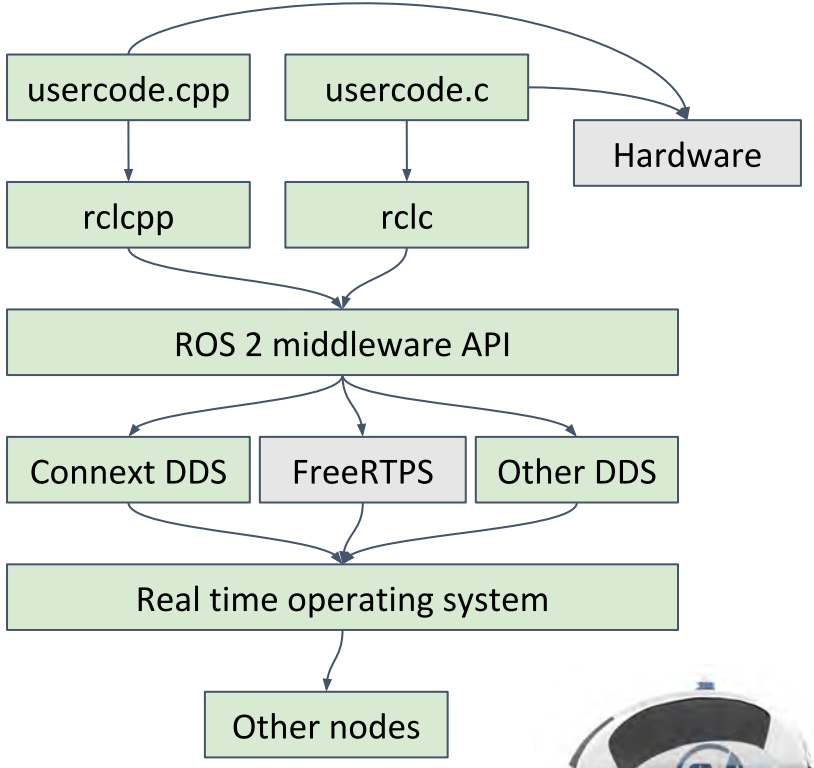
ROS 1: リアルタイムサポートがない

- リアルタイムは下の
  レイヤーの役割
  - Orocos
  - ros_control
- ROSソフトウェアは制御ループ
  などを気にしない
- 通信プロトコルは
  非リアルタイム
  - リアルタイムネットワークが
    あっても無意味
  - 同一マイコンでも
    リアルタイム不可能

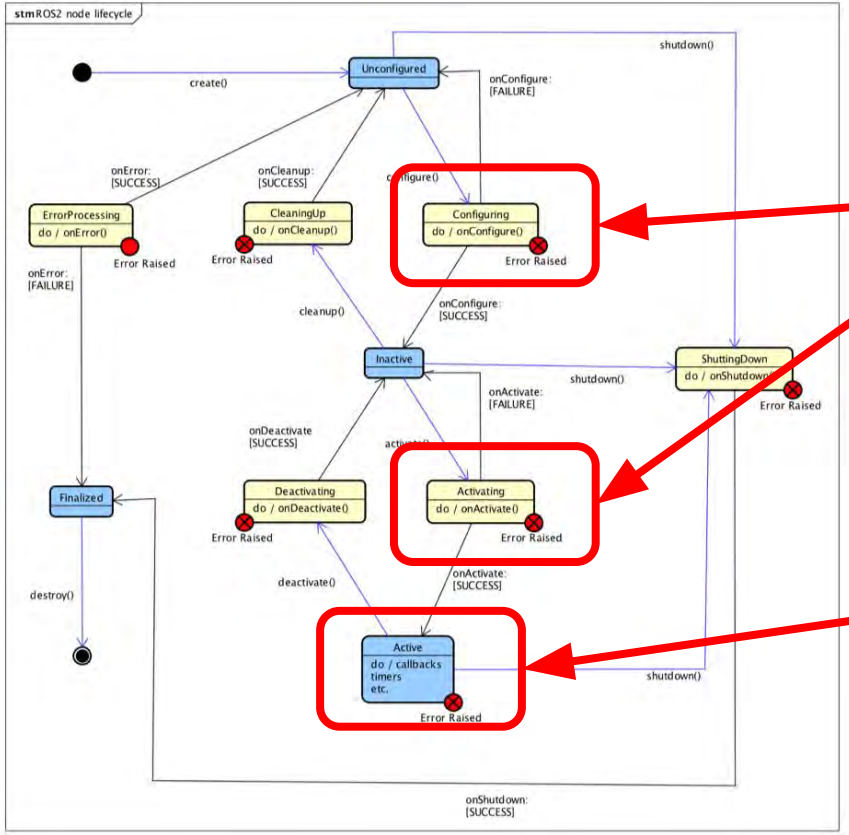


ROS 2: 全スタックがリアルタイム可能

- クライントライブラリが
  リアルタイム可能
  - （作業中）
- ミドルウェアが
  リアルタイム可能
- ハードウェア制御は純粋ノードで可能



ノードライフサイクル – ROS 2



リソース獲得などの非リアルタイムコード

リアルタイム可能決定的な動作

ROS 1: 組み込みは不可能

- PR2のユースケースは最高のCPUと大量のメモリー
- ROS 1の依存するソフトウェアは大量
  - 特にOS
- 依存ソフトウェア以外のリソース利用が大きすぎ

（近年可能になったけど理想ではない）

組み込みROS

- 組み込みマイコンはメインユースケースの一つ
  - ローパワー
  - ローメモリ
  - arm32
- DDS-XRCEプロトコルサポート
- （開発中のところはある）
- 興味がある方：

eSOL RENESAS

	"small" 32-bit MCU	"big" 32-bit MCU
Core	ARM Cortex-M0	ARM Cortex-M7
Speed	48 Mhz	300 Mhz
RAM	32 KB	384 KB
Flash	256 KB	2048 KB
Cost @ 1K units	$2	$10
Comms	USB FS	Ethernet, USB HS

Open Source Robotics Foundation

認証可能なROSへ

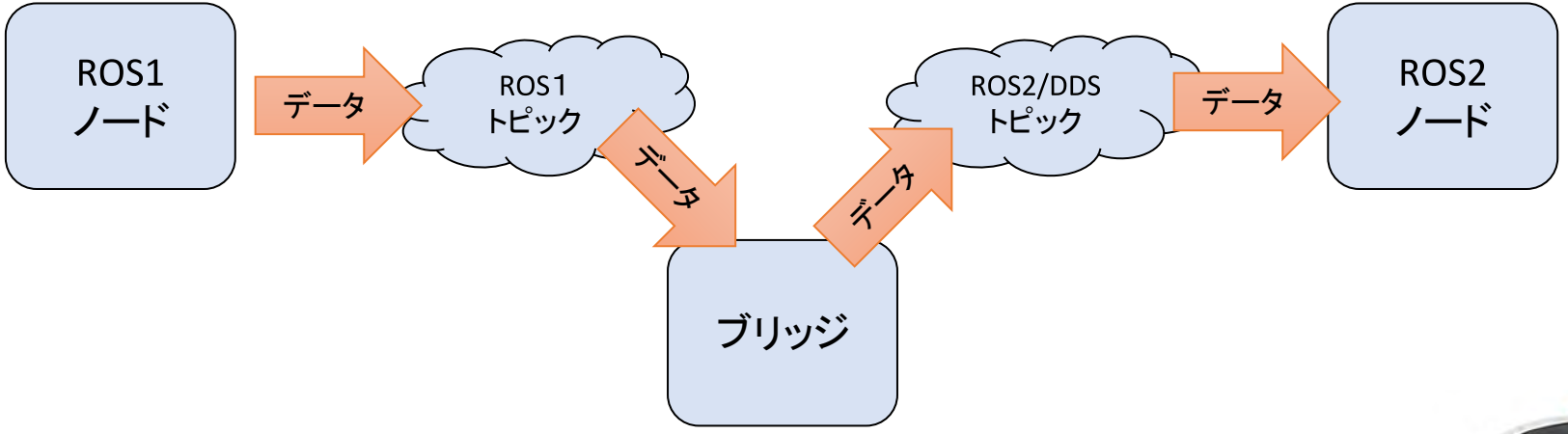
- 開発プロセスが追跡できたから認証可能
- アーキテクチャーは認証を支援
  - 必要な部分のみ利用し、それを認証する
- MISRA-C/MISRA-C++に合致を目的に
- 静的システム作成可能
  - 完全静的ROS2は開発中（Apex.io）
- セキュリティは基礎からサポートあり
  - DDS-Security規格に合致

ROS 1とROS 2の差

- 新しい通信プロトコル
- 通信プロトコルの実装は入れ替え可能
- QoSが管理可能
- ライブラリアーキテクチャの更新
- 全スタックがリアルタイム可能
- ノード構造の更新
- OSサポートの拡大
- 開発プロセスの記録
- 安全認証取得可能

ROS 1とインタラクション

- ブリッジによりROS 1のシステムとROS 2のシステムが通信可能



今あるAutowareとその問題

- 研究世界から来た
  - 品質に改善の余地がある
  - テストを強化すべき
  - ドキュメントの整備はこれから
- ROS 1を利用
  - 非リアルタイム、セキュリティに懸念、安全性は十分には担保されていない
- Autowareでプロトタイピング：🙂 → 😀 → 😃 → 😄 → 🥳
- Autowareで製品販売：😕もしくは🤕もしくは👮🚓⚖️

- 基本的にROS 1と同じ立場

必要なAutoware

- テストされた品質が高いソフトウェア
- 認証向き
  - 開発プロセスを管理する
  - 安全標準（ISO 26262、MISRA等）に可能な限り従う
  - デザインを検証する（形式手法等）
  - 上記を果たしながら、オープンコミュニティを潰さない

- 基本的にROS 2と同じ立場

Autoware.AIとAutoware.Auto

Autoware.AI

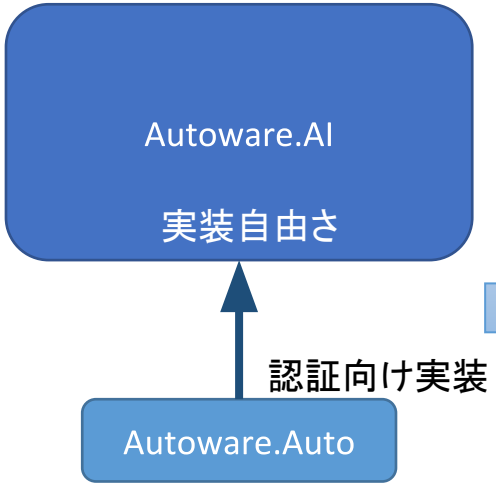
- 現在のAutowareの改善
- 様々なベストプラクティスの導入で開発プロセスの改良
- テストの追加
- ドキュメント作成
- ROS 1、ROS 2ベース

Autoware.Auto

- ゼロから再開発
- 認証向き開発プロセス
- デザイン作成、検証
- テストファスト開発（TDD）
- 開発記録作成
- ROS 2ベース
  - **ROS 2で上記の改善点が利用したい**

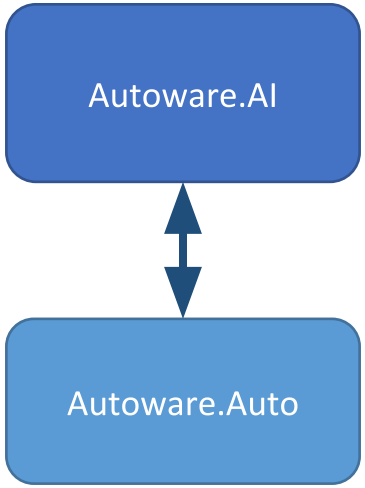
プロジェクトの合流

現在



- .Autoに無いフィーチャーが.AIから利用
- .AIはスタンドアローンで利用可能

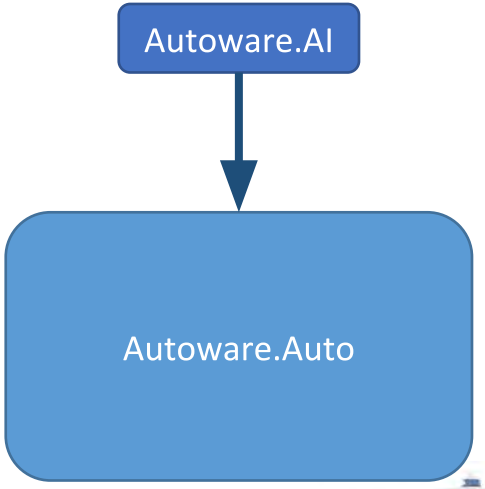
途中



- お互いにフィーチャーを利用

将来

- .AIは.Autoの別名
- .AIは認証向けでは無い



https://roscon.jp