TOPPERS
Toyohashi OPen Platform for Embedded Real-time Systems


# ETロボコン向け 箱庭技術とその仕組み

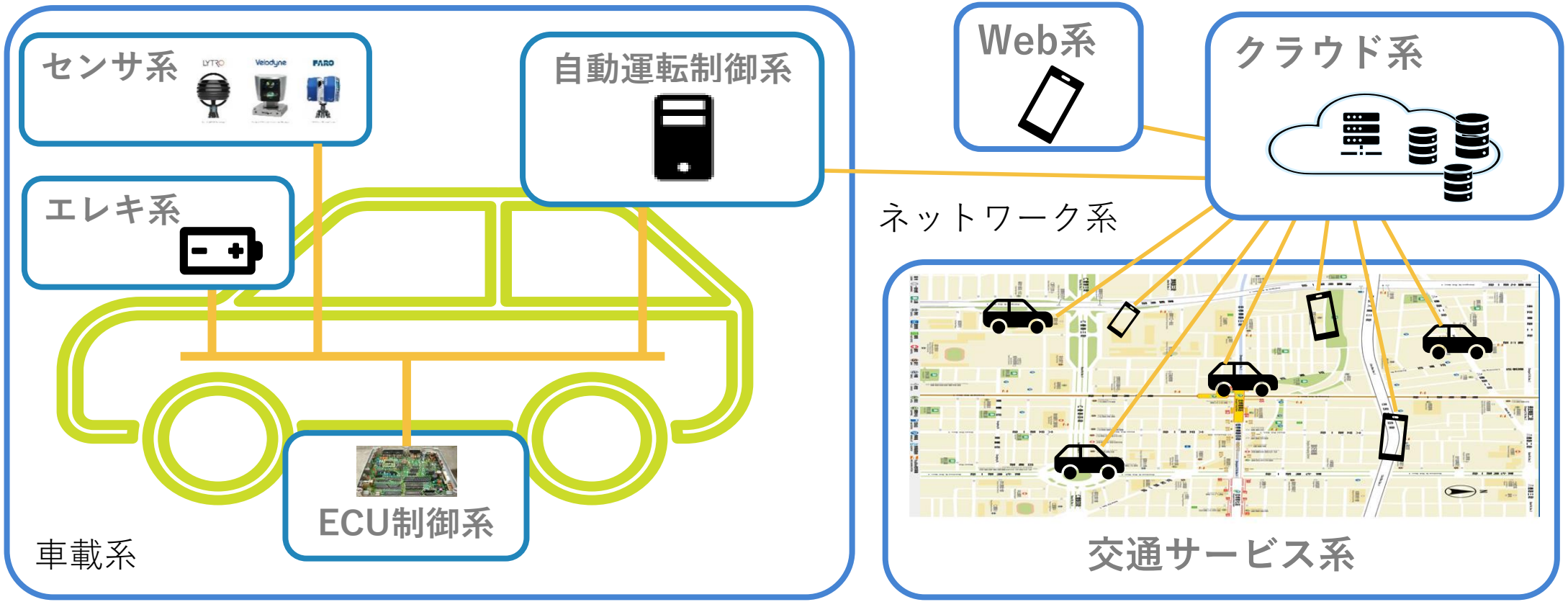
株式会社 永和システムマネジメント　森　崇

# アジェンダ

1. 箱庭とは
2. ETロボコンで利用されている箱庭の要素技術
3. 箱庭要素技術とその仕組み
4. ノウハウ集
5. 箱庭WGへのお誘い

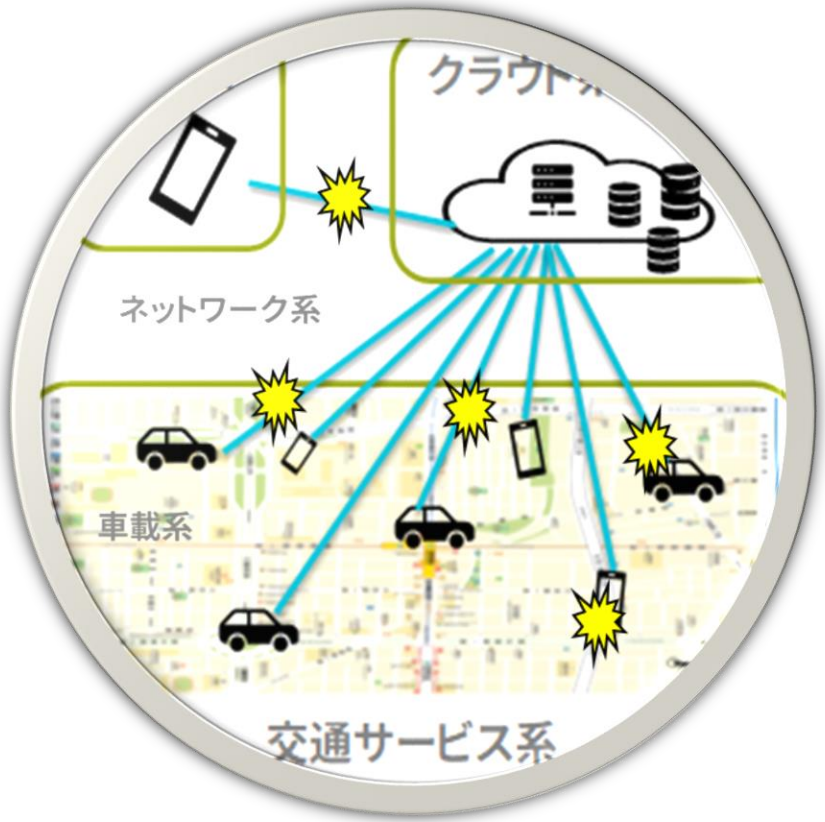
# 箱庭とは：背景

**例えば自動運転システムでは,,,**
**様々な分野の技術領域を横断 している！**



# 箱庭とは：課題

- 問題発生経路の複雑化
  - 全体結合しないと見えない問題が多数潜んでいる
  - 様々な機器間の整合性を取れない
- 原因調査の複雑化
  - どこで何がおこっているのか調査困難
  - そもそもデバッグすること自体が難しい
- 実証実験のコスト増
  - 実証実験は手軽に実施できない
  - 各分野のエンジニアの総動員
  - 手間，時間，費用がかかる

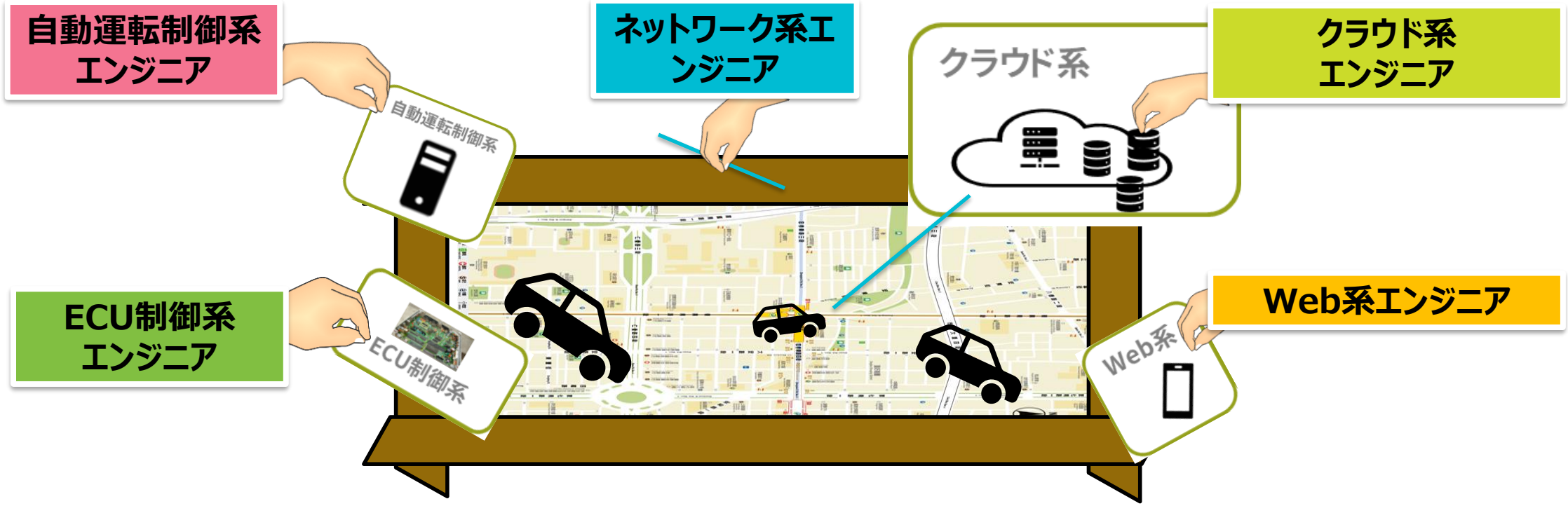


# 『箱庭』の狙いとコンセプト

**箱の中に，様々なモノをみんなの好みで配置して，いろいろ試せる！**

**・仮想環境上(箱庭)でIoT/自動運転システムを開発する**

⇒ 各分野のソフトウェアを持ち寄って，机上で全体結合＆実証実験！

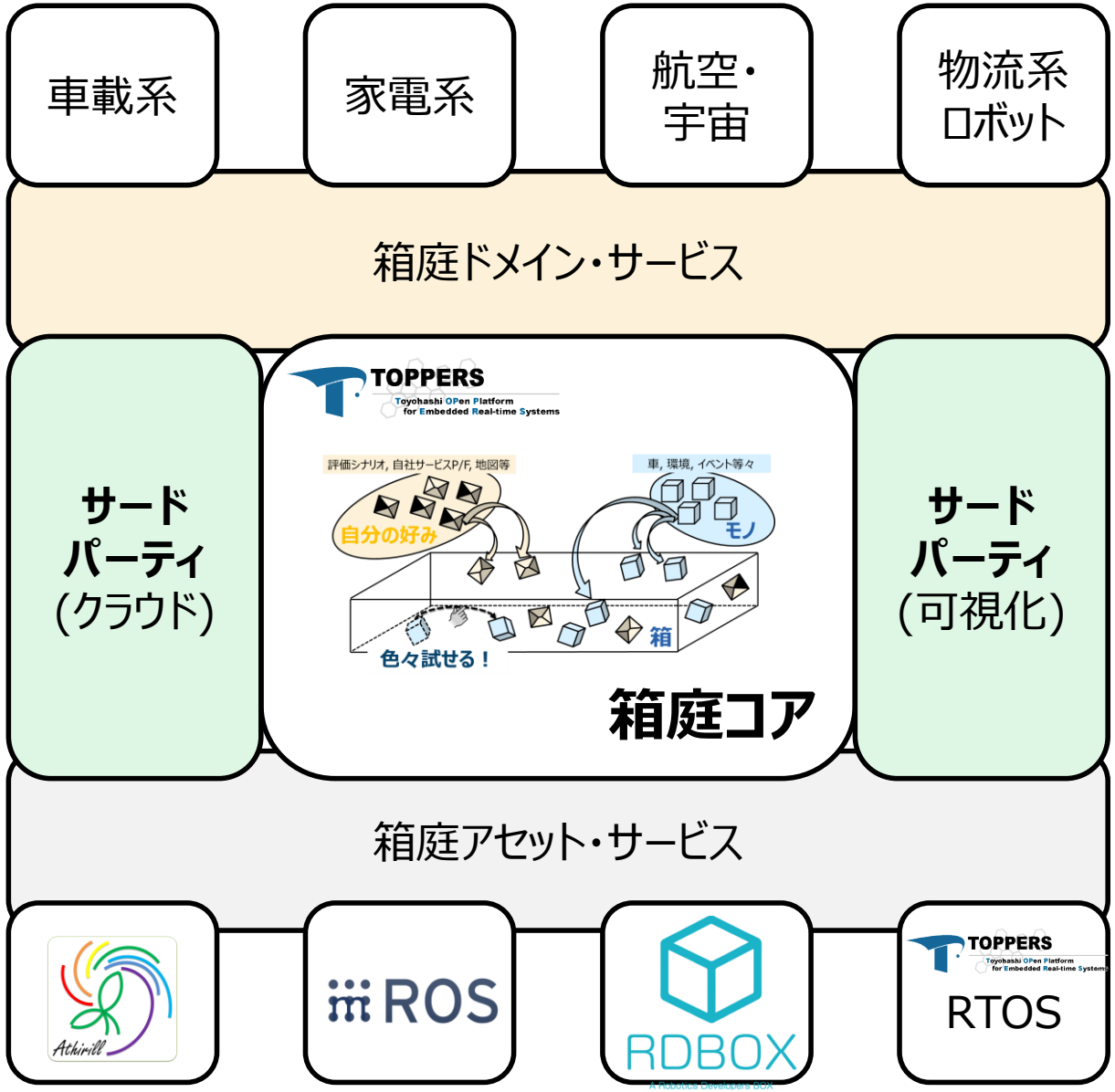


# 箱庭コンセプトとアーキテクチャ

■箱庭ドメイン・サービス
　様々な分野への適応を目指す

■箱庭コア
　箱庭固有のシミュレーション技術をコア技術化
**Hakoniwa Engine**
■サードパーティ
　既存のサードパーティ製で出来ていることは積極利用

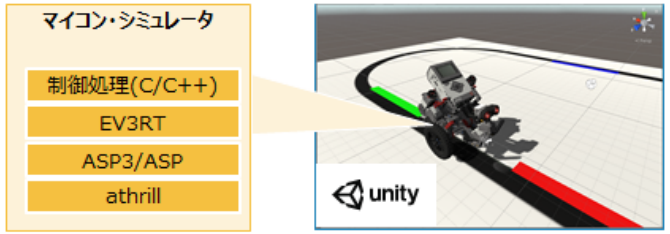
■箱庭アセット・サービス
　シミュレーション内の登場物を箱庭アセット化し，アセット数拡充を目指す



# 箱庭プロトタイプモデル

## ３つのプロトタイプモデルを構築しています！
### (目的：箱庭コンセプトの実現/技術研鑽)

### A：単体ロボット(ETロボコン)向けシミュレータ




ETロボコンを題材として構築

**技術研鑽視点での狙い：**
・物理シミュレータとマイコンシミュレータ間の連携方法の検討
・異なるシミュレータ間の時間同期の検討

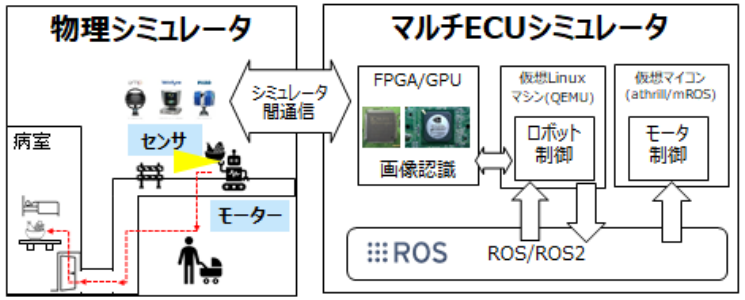
**その他の狙い：**
・ETロボコンユーザ層に箱庭を広める（広報活動）

Unityパッケージの設計と作成にあたっては，
宝塚大学 東京メディア芸術学部 古岡章夫准教授
および学部生の杉﨑涼志さん，木村明美さん，
千葉純平さんにご協力いただきました。

HackEVのUnityアセットは，ETロボコン実行委員会より提供いただいたデータを基に作成しています．
実行委員会の皆さまに深く感謝いたします．
ただし本アセットはETロボコンの本番環境とは異なりますので，大会に参加予定の方はご注意ください．
また，本アセットは，個人利用または教育利用に限定してご利用ください．



### B：ROS･マルチECU向けシミュレータ

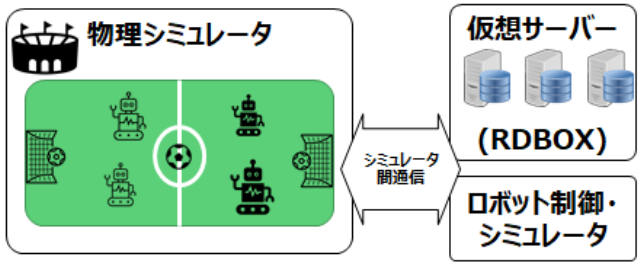



**技術研鑽視点での狙い：**
・マルチECU/FPGA/GPU間の連携方法検討(シミュレーション時間同期等)
・箱庭アセット間の通信可視化方法の検討(ROS/ROS2連携含む)
・箱庭アセットの仕組み検討

**その他の狙い：**
・ROSユーザ層に箱庭を広める（広報活動）
・その他チャレンジ(つくばチャレンジ/FPGAデザインコンテスト)



### C：ロボット間協調動作向けシミュレータ




**技術研鑽視点での狙い：**
・クラウド連携方法検討
・ロボット間の連携方法検討(より複雑なロボットの動き/干渉に挑戦)
・箱庭アセットを増やす仕組みの検討

**その他の狙い：**
・RDBOX連携(開発支援仮想環境としての箱庭の実績作り)
・RDBOXユーザ層に箱庭を広める（広報活動/ROSCon JP 参加）

# アジェンダ

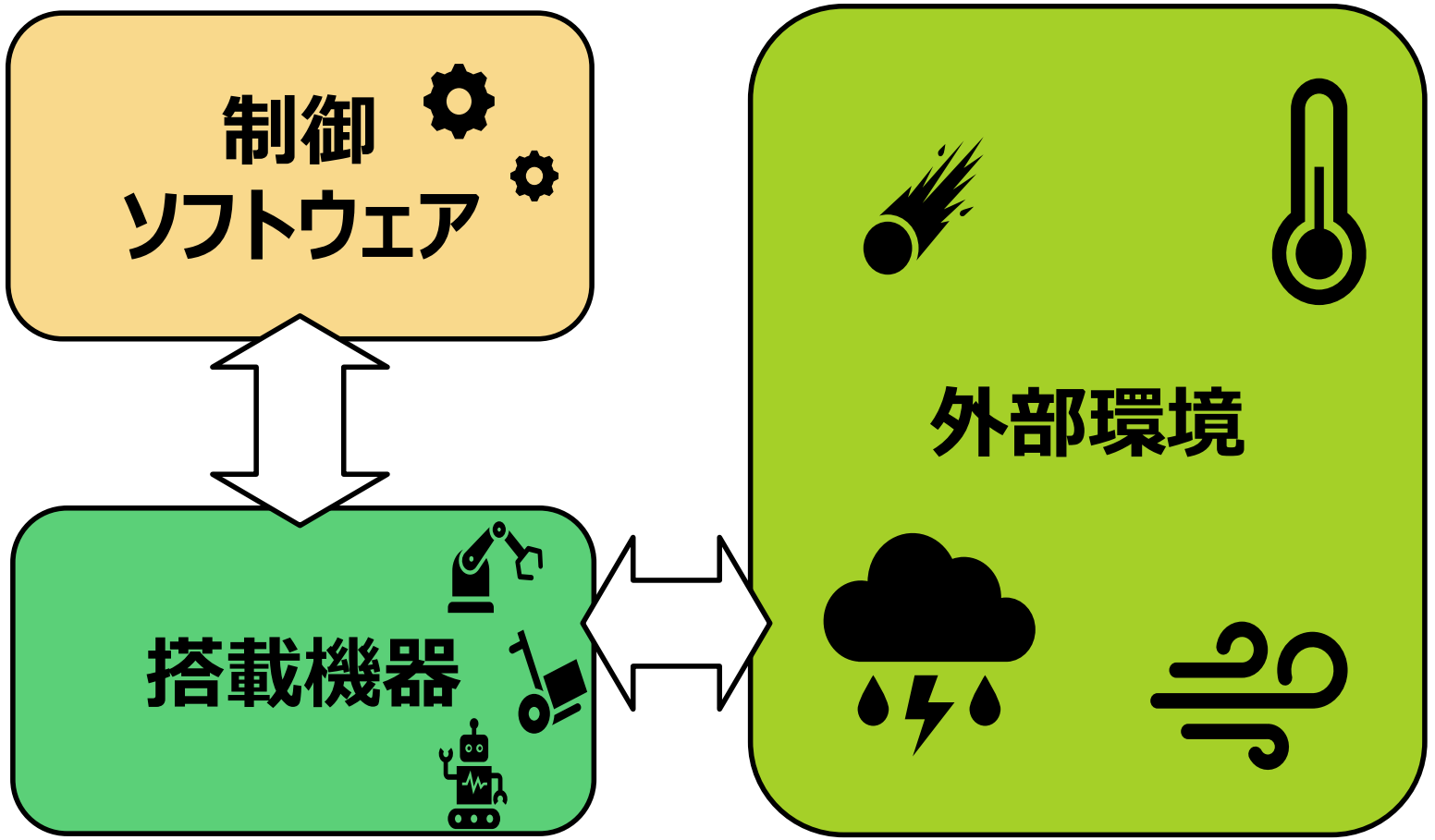
1. 箱庭とは
2. **ETロボコンで利用されている箱庭の要素技術**
3. 箱庭要素技術とその仕組み
4. ノウハウ集
5. 箱庭WGへのお誘い

# ETロボコンで利用されている箱庭の要素技術

- ETロボコンの構造
- 何を目的としたシミュレータなのか
- 精度/コスト/うれしさのバランス
- ETロボコンシミュレータの構造

# 組込みシステムを抽象化し３分類すると

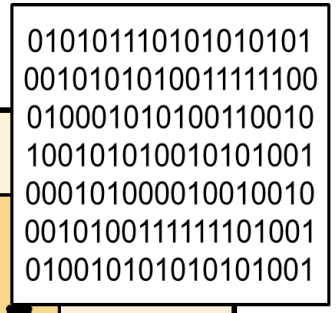
- 制御ソフトウェア
- 搭載機器
- 外部環境

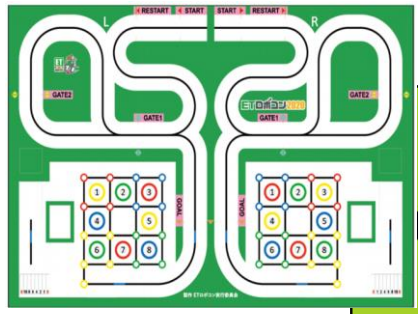


# ETロボコンの構造

走行体制御ソフトウェア

制御ソフトウェア



外部環境

EV3RT/HRP3

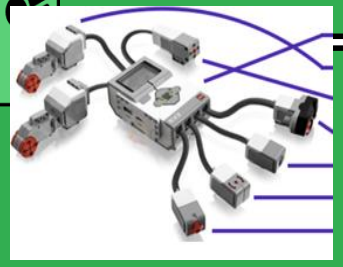
走行環境

マイコン(ARM)

搭載機器

走行体(EV3)

センサ・アクチュエータ

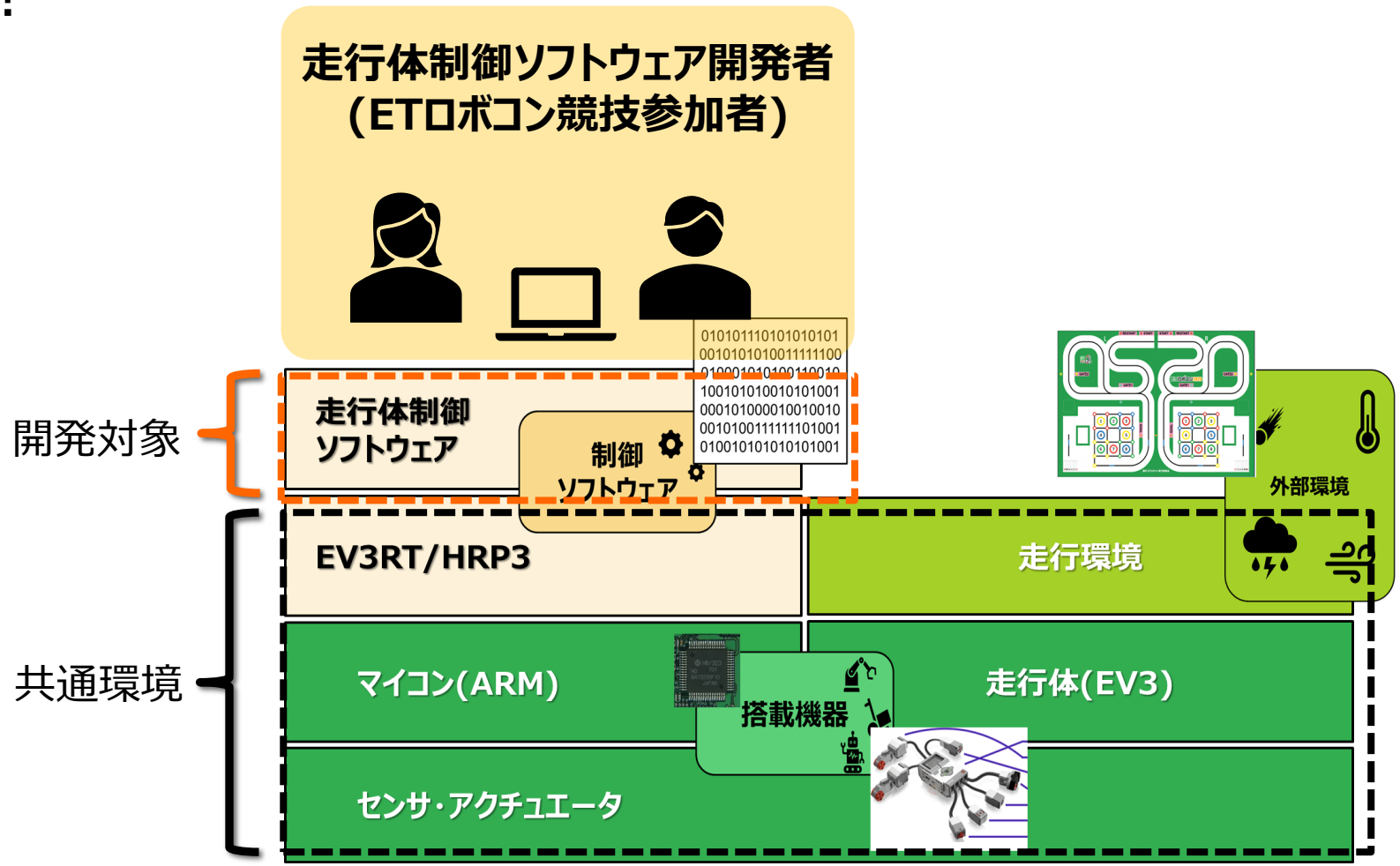


# ETロボコンで利用されている箱庭の要素技術

- ETロボコンの構造
- **何を目的としたシミュレータなのか**
- 精度/コスト/うれしさのバランス
- ETロボコンシミュレータの構造

# なにを目的としたシミュレータなのか(1/2)

- 誰をハッピーにしたいのか？



# なにを目的としたシミュレータなのか(2/2)

- どういう課題を解決したいのか？

## Cost

- 実機/コース購入費用
  ⇒実機/コースの奪い合い！
- 実験するには広いスペースが必要
  ⇒実験場の準備/移動しないといけない！

## Quality

- ロボット制御/RTOSの知識不足
  ⇒初心者だと普通に走らせるまでが大変！
- 光センサの外乱/コースの微妙なズレ
  ⇒物理的な外乱系は狙ってテストできない！

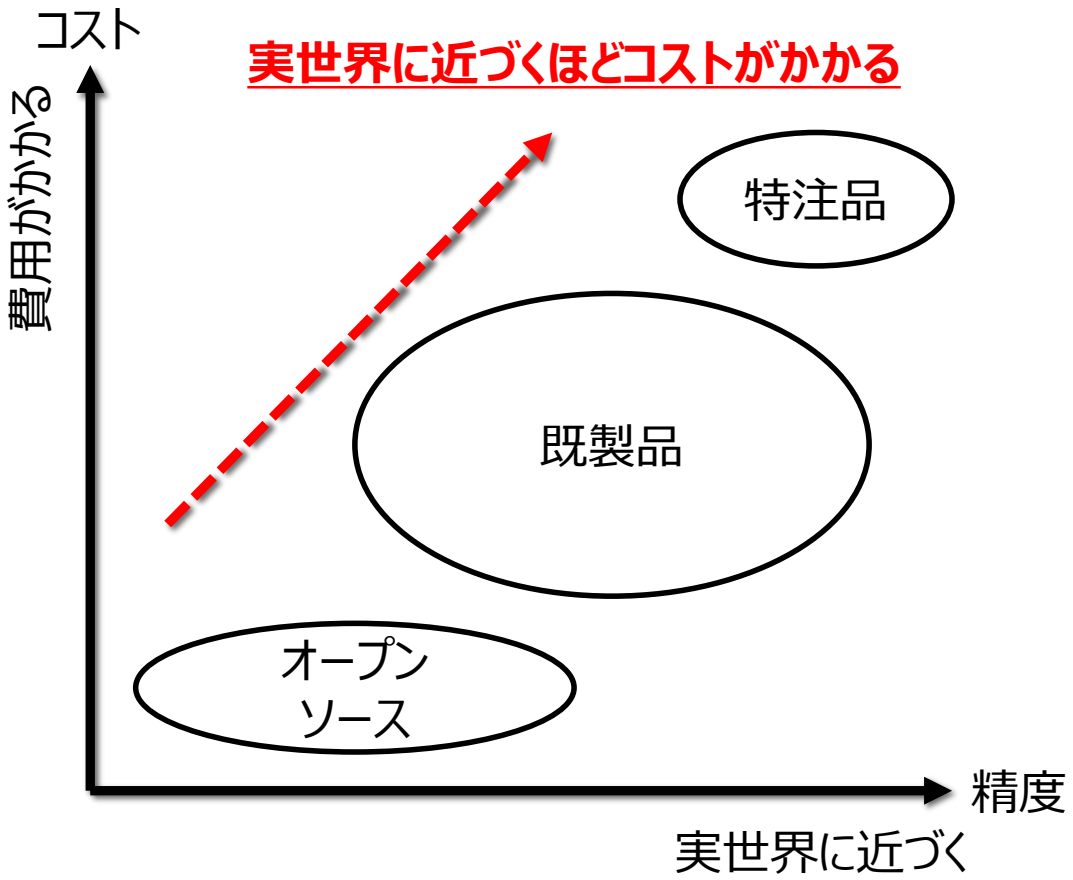
## Delivery

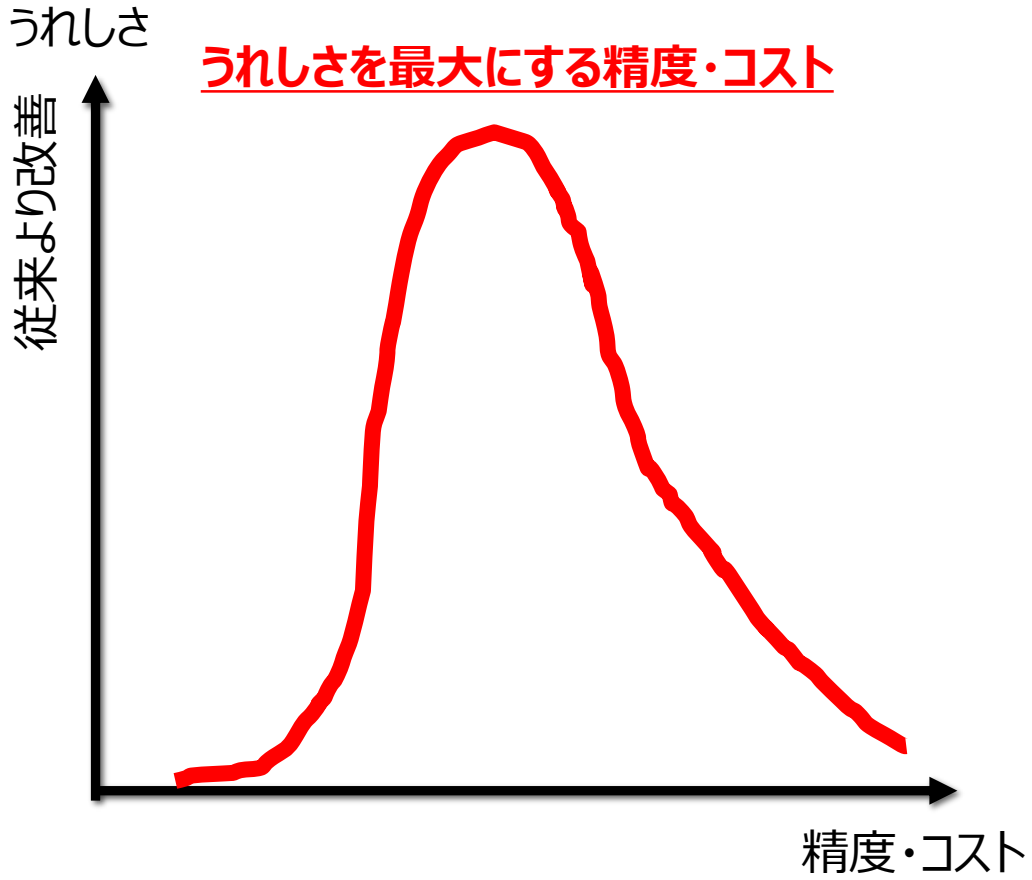
- デプロイ/デバッグの手間
  ⇒ひたすら時間をかけてやる！
- 特定シーン/条件のテストの手間
  ⇒ひたすら同じシーンを繰り返す/試行錯誤
- バッテリ切れ，実機故障
  ⇒そもそもテストできない！

# ETロボコンで利用されている箱庭の要素技術

- ETロボコンの構造
- 何を目的としたシミュレータなのか
- **精度/コスト/うれしさのバランス**
- ETロボコンシミュレータの構造

# 精度とコストのバランス




コストを無視してシミュレータ精度を実世界に近づけるとハッピーになれるか？

# 箱庭WG活動ではじめたこと

- TOPPERS成果物と箱庭要素技術を組み合わせる
  - 決意表明から約６か月で動くものができた！


Qiita　キーワードを入力

## 全体進捗状況

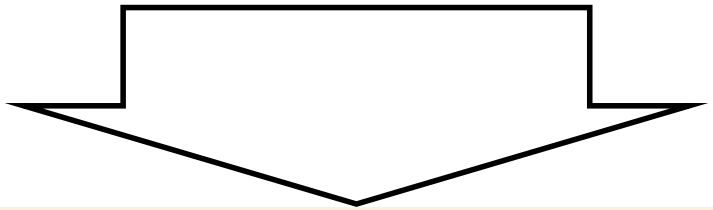
- 2019/04/20：決意表明しました.
- 2019/05/01：物理シミュレータの勉強した内容した.
- 2019/08/05：athrill2/mROS/Unity連携した
- 2019/08/24：hrp2の移植開始した
- 2019/10/13: EV3RTの移植調査を始めた
- 2019/10/20: EV3RTの移植が進み始めた(LED，ボタン，センサ系が動くようになった)
- 2019/11/04: EV3RTのサンプルコード(ラントレース)を移植してUnity上で動くようにした

### ETロボコン・シミュレータ構成

ETロボコンのシミュレータ構成としては，下図のように２つのシミュレータを連携させています.

# 精度・コストのバランスから諦めたこと

- 実機用のバイナリをそのままシミュレータで動作させること
  - EV3RTが利用している各種デバイスをすべて仮想化するには時間が掛かりすぎる
    ⇒EV3RTのAPIレベルの再現ができることに注力
- 実機用のCPUの仮想化
  - EV3はARM版ですが，Athrillを前提にしていたのでV850版にしました
- 実機用のRTOS(HRP3カーネル)で動作させること
  - TOPPERS第３世代カーネルの中で Athrill が動作するのは ASP3カーネル だけだったので…

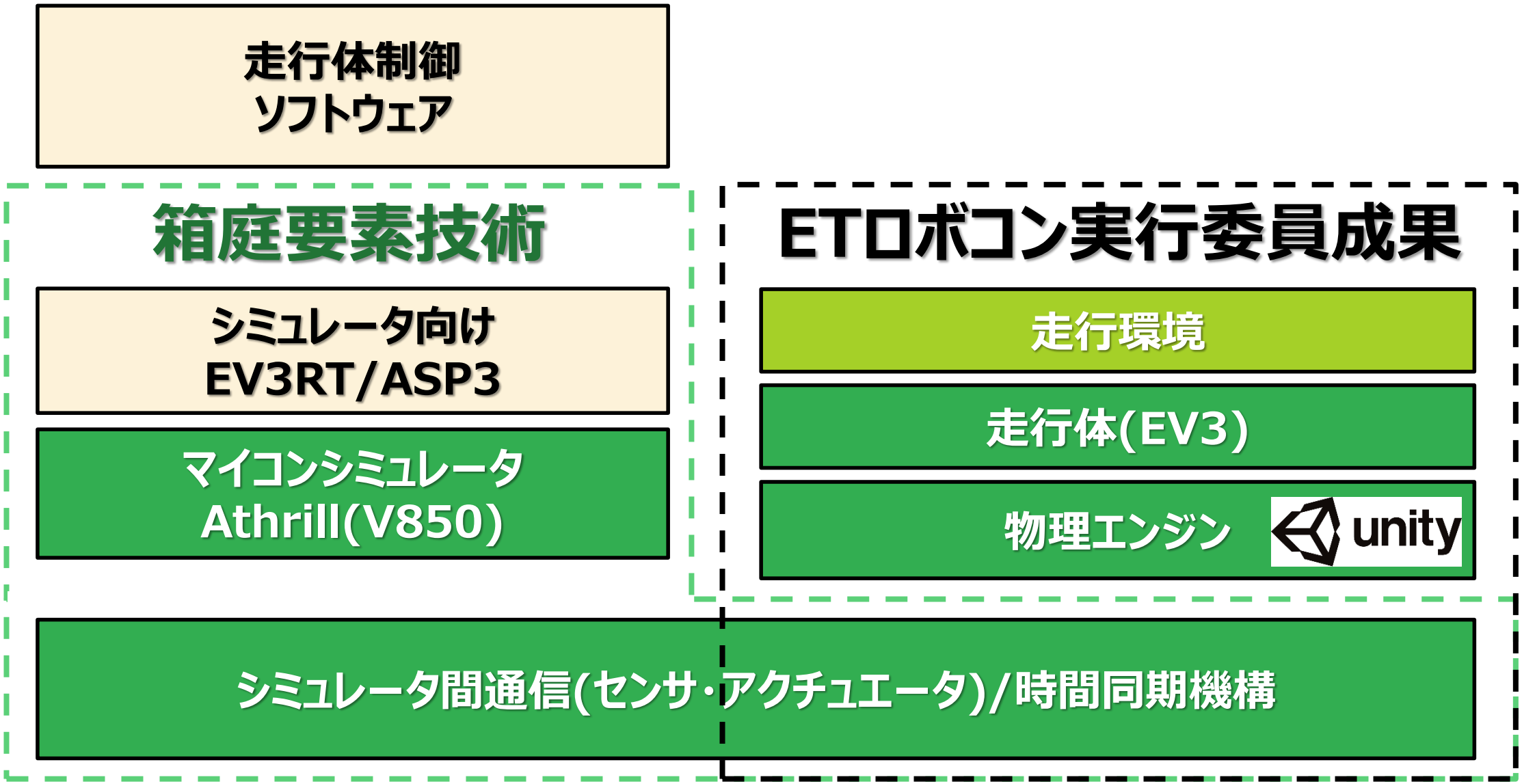


**ETロボコン競技参加者が作成した**
**走行体制御アプリケーション(C/C++)を**
**そのままシミュレーションできる**

# ETロボコンで利用されている箱庭の要素技術

- ETロボコンの構造
- 何を目的としたシミュレータなのか
- 精度/コスト/うれしさのバランス
- **ETロボコンシミュレータの構造**

# ETロボコンシミュレータの構造



# アジェンダ

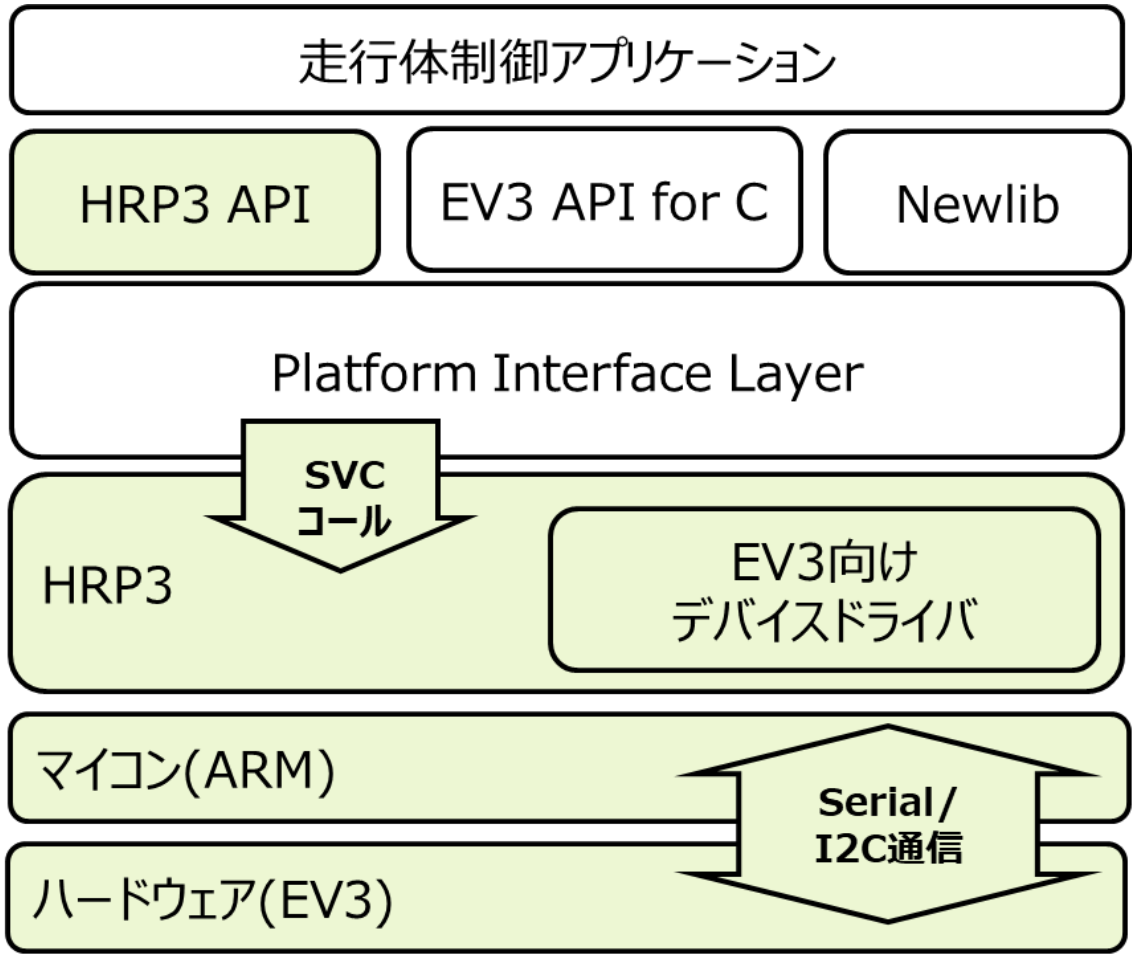
1. 箱庭とは
2. ETロボコンで利用されている箱庭の要素技術
3. **箱庭要素技術とその仕組み**
4. ノウハウ集
5. 箱庭WGへのお誘い

# 箱庭技術とその仕組み

- シミュレータ向けEV3RT/ASP3
- マイコンシミュレータAthrill
- シミュレータ間通信/時間同期機構

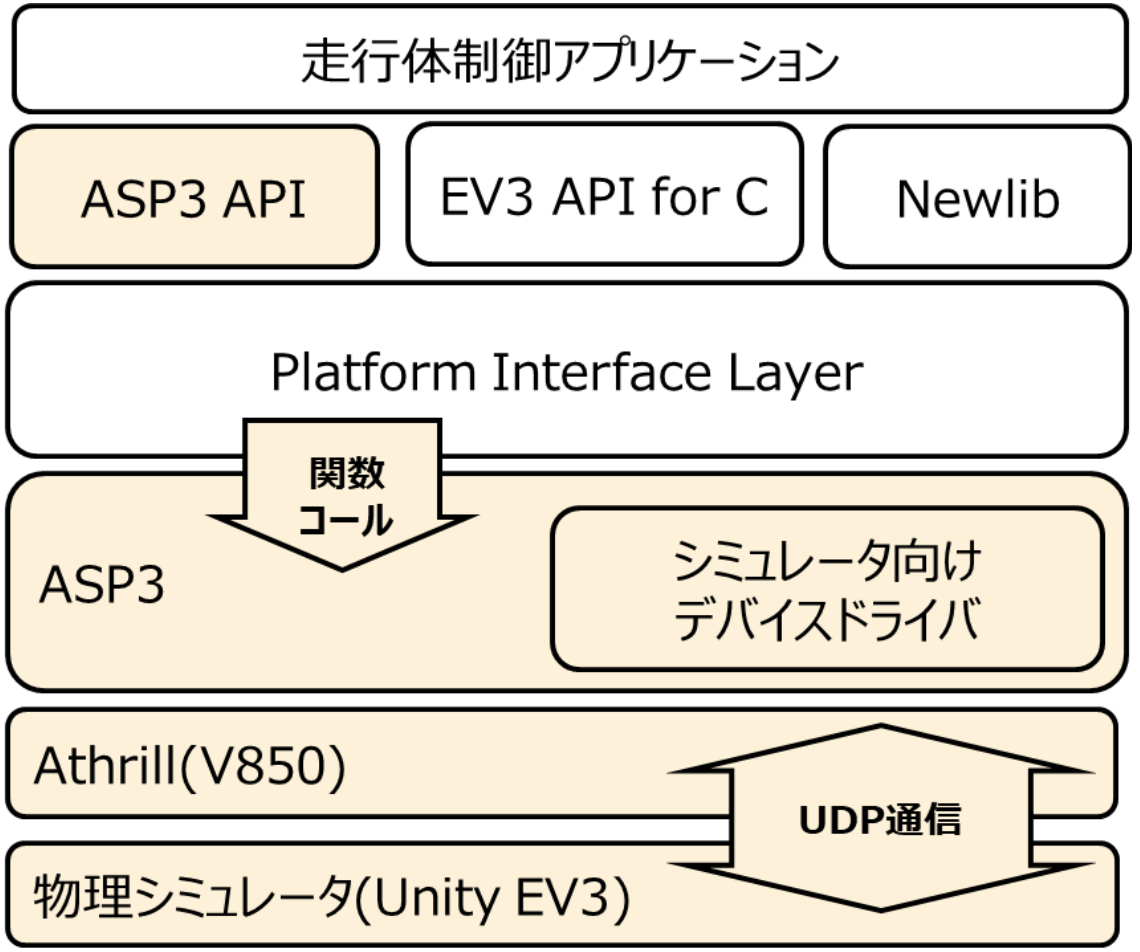
# シミュレータ向けEV3RT/ASP3

- 本番環境
  - シミュレータと差分がある機能を色分け



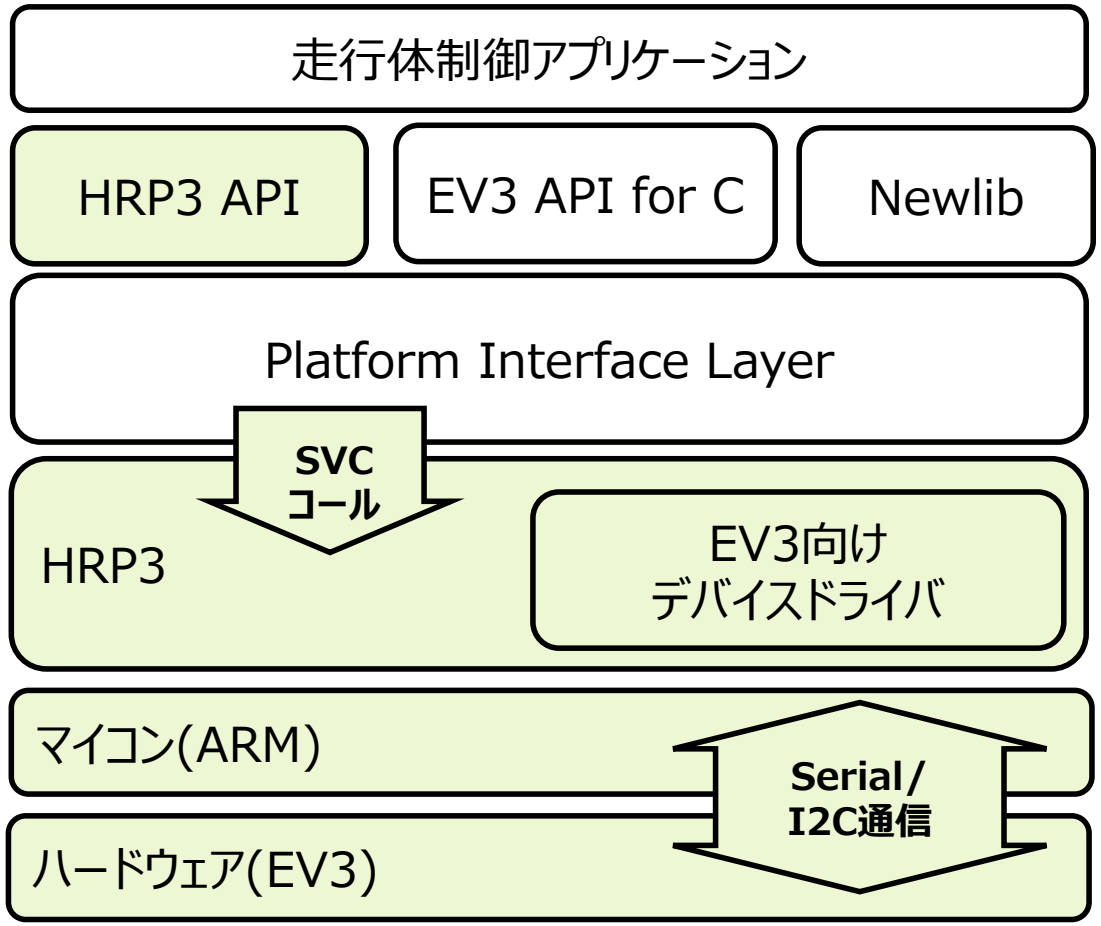
# シミュレータ向けEV3RT/ASP3

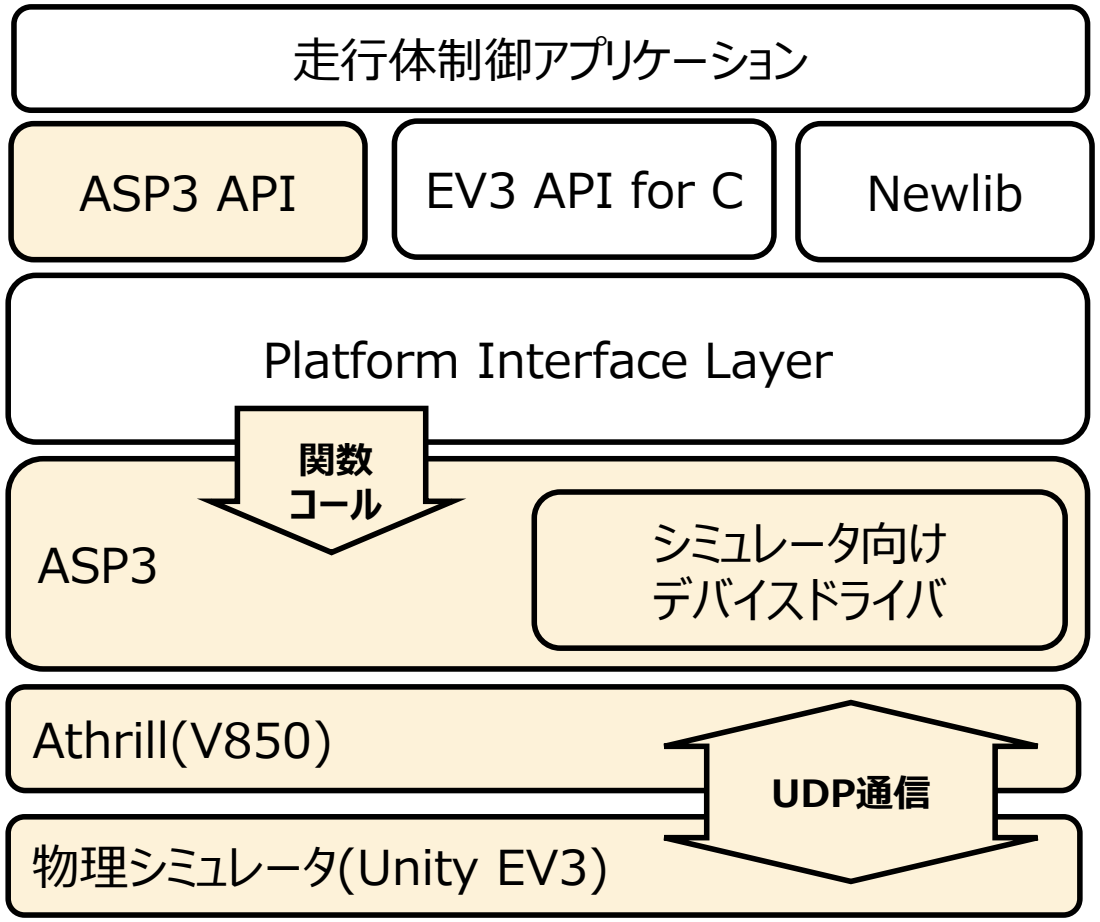
- シミュレータ
  - 本番環境と差分がある機能を色分け



# シミュレータ向けEV3RT/ASP3

- 走行体制御アプリケーションが，TOPPERS第3世代カーネル（ITRON系）の基本仕様(ASP3カーネル仕様)のAPIを使用していれば，本番環境でそのまま利用できます.
  - シミュレータ環境では，HRP3カーネル仕様のメモリ保護機能APIは利用できません.



# 箱庭技術とその仕組み

- シミュレータ向けEV3RT/ASP3
- **マイコンシミュレータAthrill**
- シミュレータ間通信/時間同期機構

# マイコンシミュレータ Athrill

- マイコンシミュレータとは
  1. 命令セットシミュレータ
  2. コンピュータのシミュレータのモデルのひとつで，
     命令セットレベルのシミュレーションを行うものである

  ※参照元：ウィキペディア

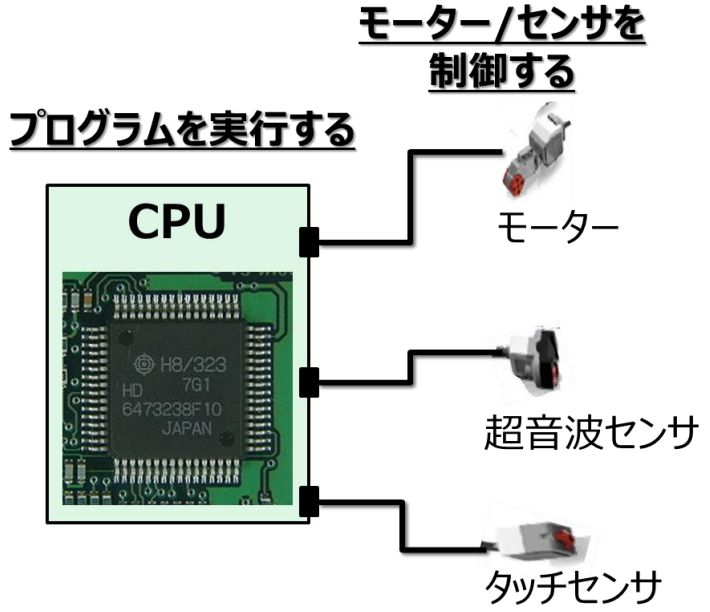
- Athrillでできること
  1. 命令セット(バイナリファイル)を実行できます
  2. Unityと通信できます
     - Unityで作成したモーターを動かすことができます
     - Unityで作成したセンサで障害物を検出できます

週末作った！

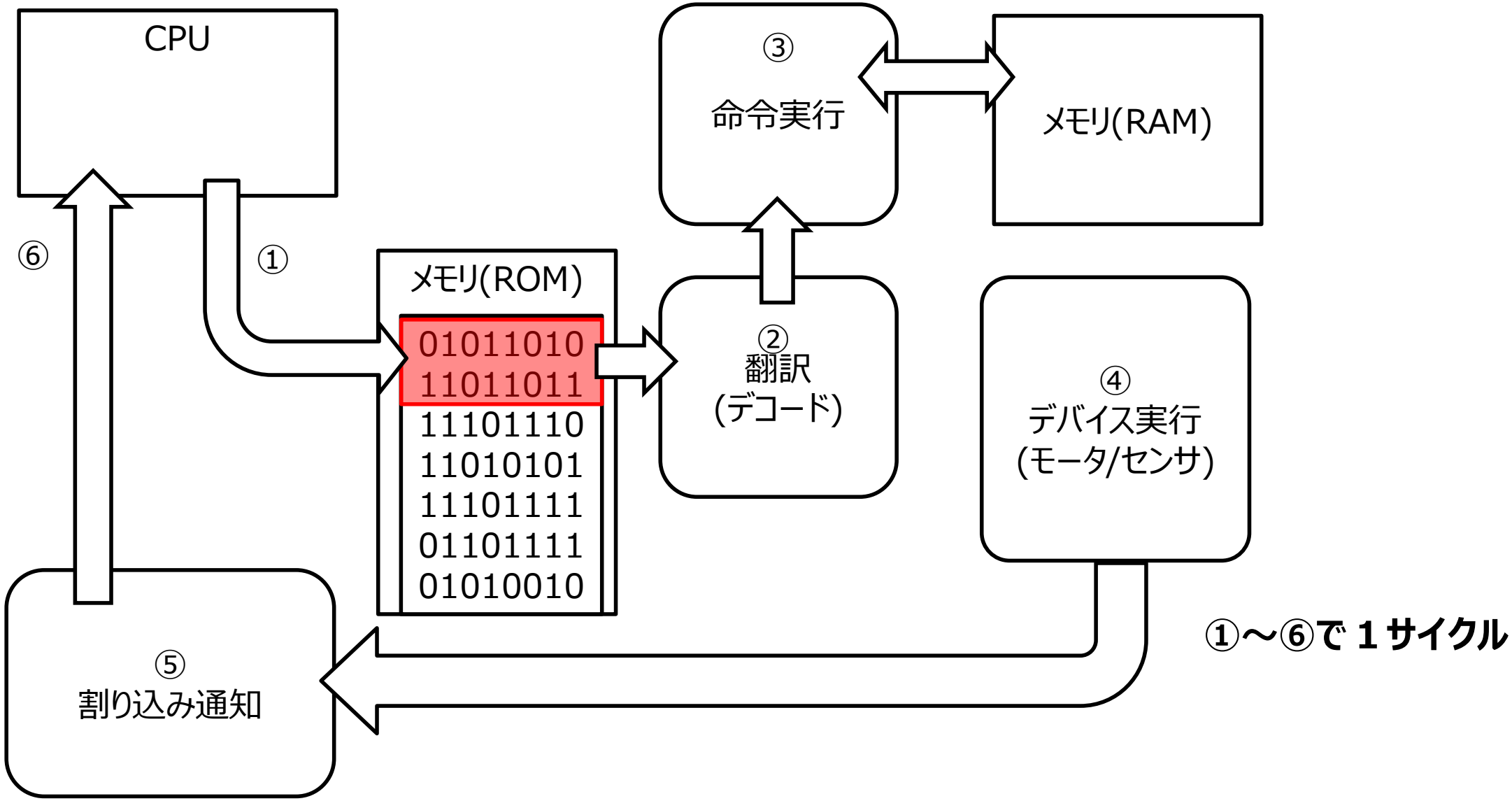
**機械が理解できる言語**

バイナリファイル(asp)

```
0101011101010101
0010101010011111100
0100010101001100 10
1001010100101010 01
0001010000100100 10
0010100111111101001
0100101010101001
```



# Athrillの処理の流れ



# Athrillの実行環境

**クロスプラットフォーム(Windows/Mac/Linux/Docker)環境で動きます**
**※おススメはWSL/Linux**

UNIX系ターミナル(Bash)

WSL/WSL2

Windows

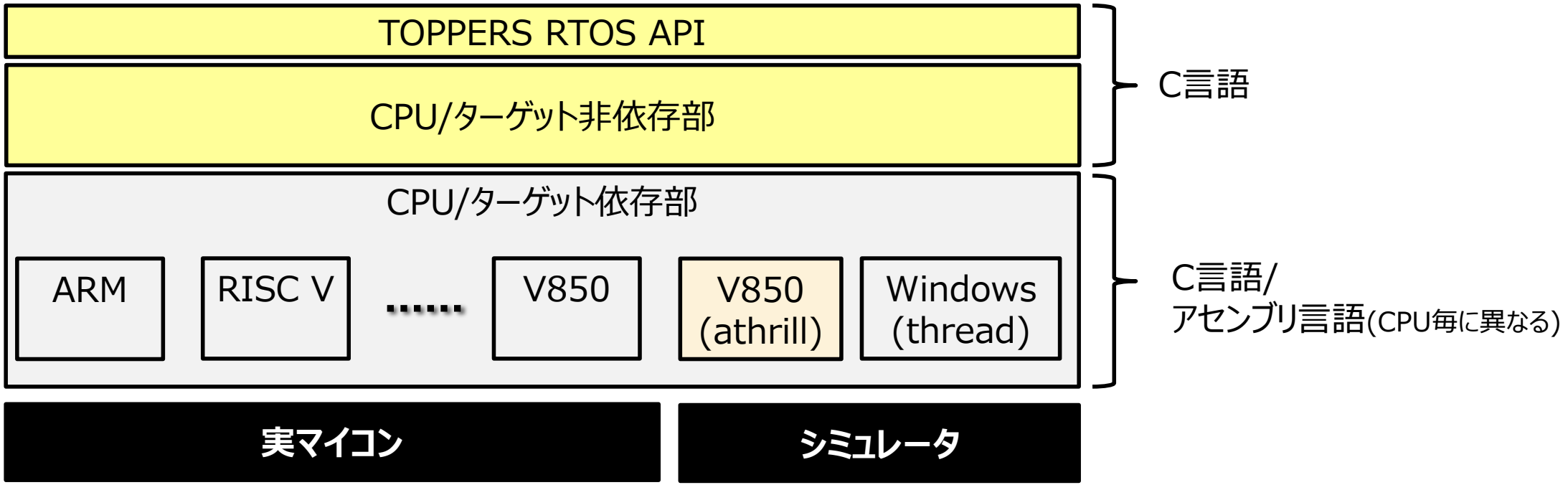

Mac


Linux


Docker

# ETロボコン向けシミュレータでAthrillを利用した理由

- ETロボコン向けのRTOSをシミュレーションするには
  - CPU/ターゲット依存部(CPU命令が多い)をPC上で実行できるようにする必要がある
    - マイコンシミュレータ(Athrill)はその選択肢の1つ.
    - CPU/ターゲット依存部をRTOS仕様に準拠してWindowsプログラムで作成することもできる
  - 開発リソースの問題で，マイコンシミュレータ Athrillを選択しました.
    - メリット　：実行環境に依存せずシミュレーションは同じ結果になる
    - デメリット：実行速度が遅い



# 箱庭技術とその仕組み

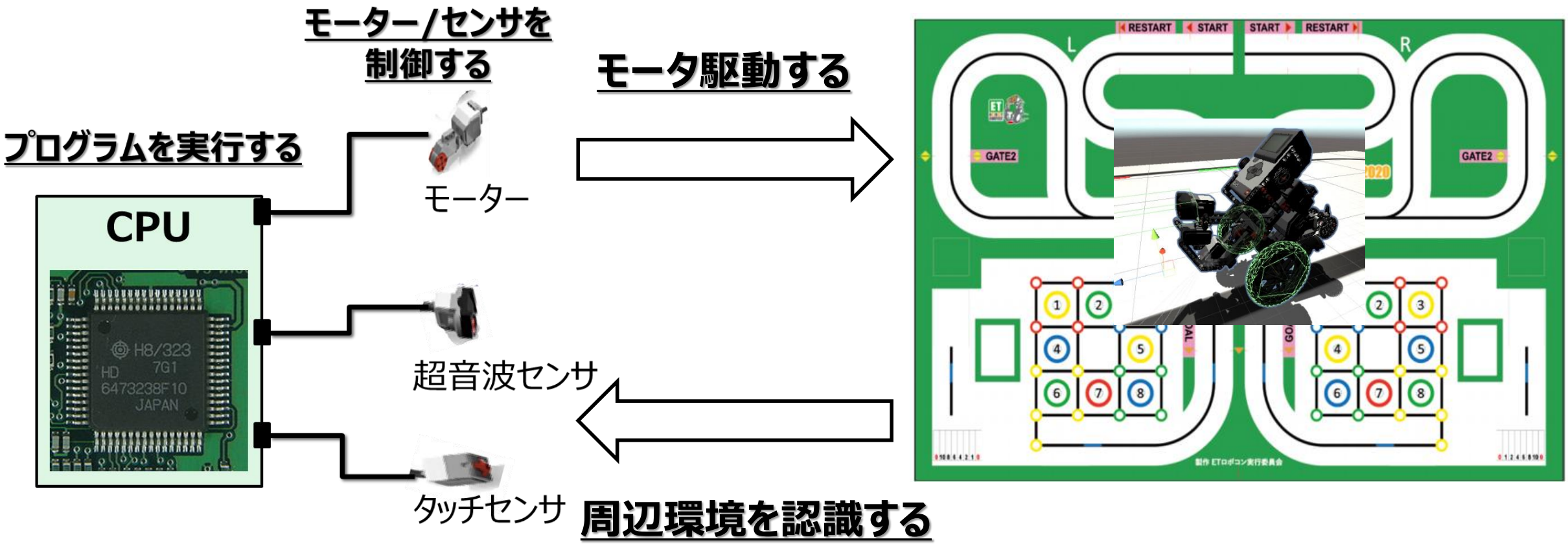
- シミュレータ向けEV3RT/ASP3
- マイコンシミュレータAthrill
- **シミュレータ間通信/時間同期機構**

# シミュレータ間通信/時間同期機構

- センサ/アクチュエータ
- シミュレーション時間と時間同期の背景
- シミュレーション時間同期機構
- ETロボコンシミュレータの通信

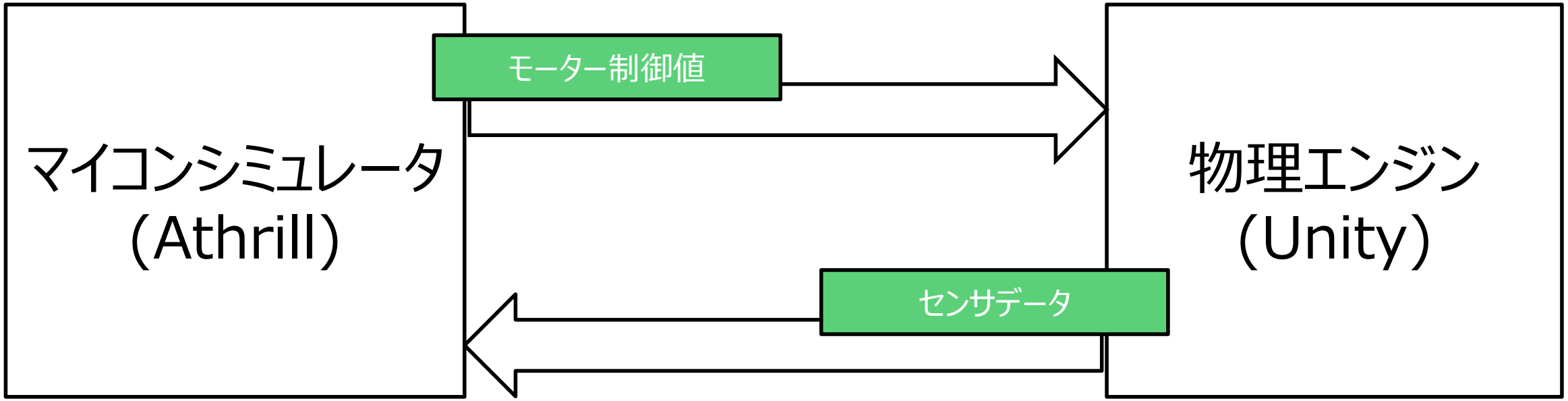
# センサ/アクチュエータ

モーター駆動すると，EV3が移動し，センサで周辺環境を認識する



# シミュレータ間の通信データ

- モーター制御値
  - マイコンシミュレータから物理エンジンに送信する
- センサデータ
  - 物理エンジンからマイコンシミュレータに送信する




# シミュレータ間通信/時間同期機構

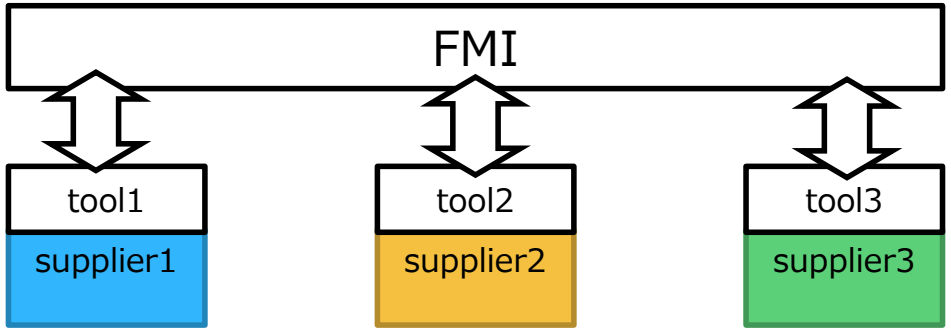
- センサ/アクチュエータ
- **シミュレーション時間と時間同期の背景**
- シミュレーション時間同期機構
- ETロボコンシミュレータの通信

# シミュレーション時間と時間同期の背景

- **箱庭の構成要素**
  - 様々なシミュレータが混在を許す環境です
- **シミュレーション時間**
  - 各シミュレータはそれぞれ固有のシミュレーション時間を持ちます(現実の時間とは異なります)
- **シミュレーション時間同期**
  - 各シミュレータが独立して動作するとシミュレーション実行タイミングがズレます
  - 箱庭環境では，これらのシミュレーション時間を同期させる方法を確立します

# 既存のシミュレータ時間同期方式

- FMI（Functional Mock-up Interface)
  - 欧州の公的プロジェクトが規格化
  - シミュレーションツールに依存しないモデル接続のための共通インターフェース



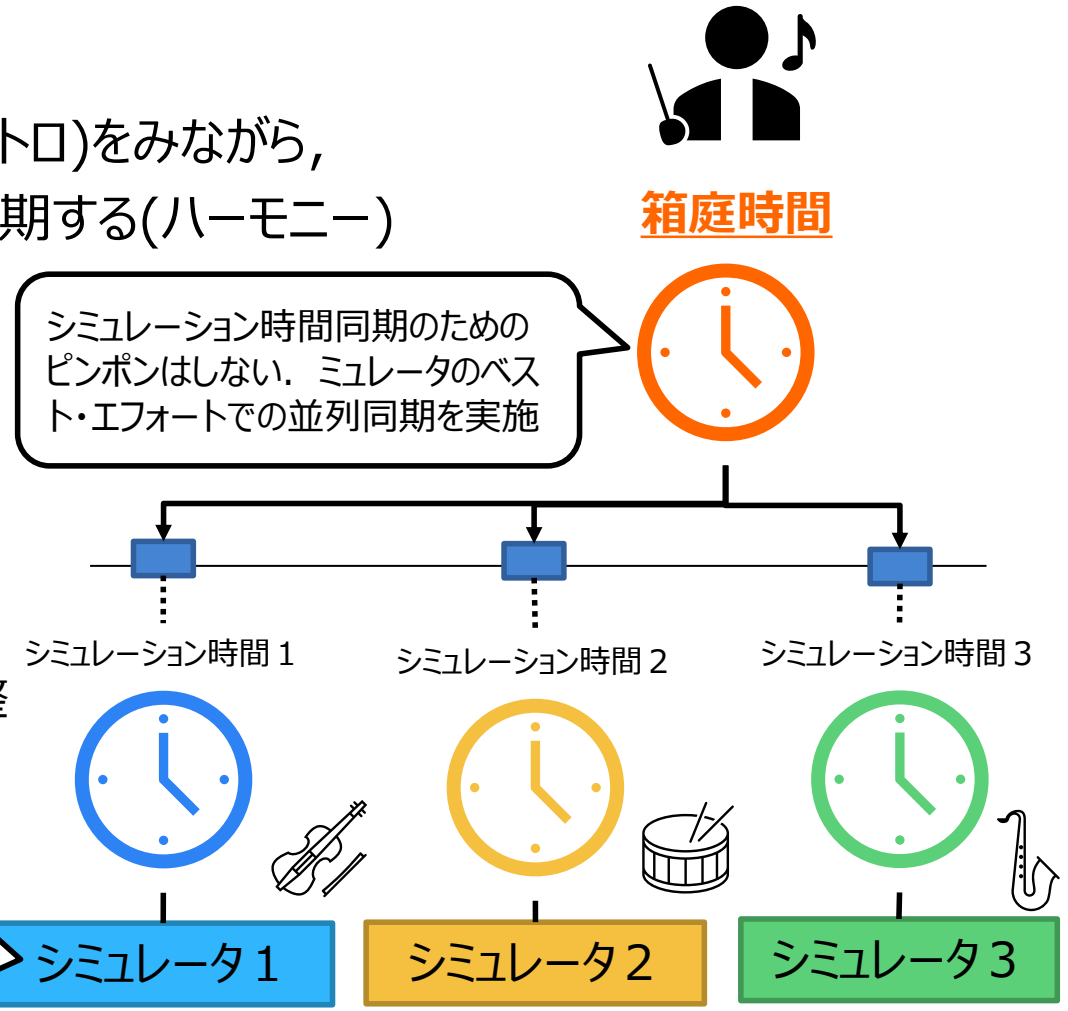

- シミュレーション時間同期方式
  - ME(Model Exchange)
    - 完全に時間同期が可能
  - CS(Co-Simulation)
    - シミュレータ間の遅延時間(固定)を許容
    - 適切な遅延時間をユーザが選択する
  - 考察
    - いずれも中央制御方式であるため，精度調整は容易であるがシステム構成要素が増えると処理オーバーヘッドが高くなると考えられる.

# シミュレータ間通信/時間同期機構

- センサ/アクチュエータ
- シミュレーション時間と時間同期の背景
- **シミュレーション時間同期機構**
- ETロボコンシミュレータの通信

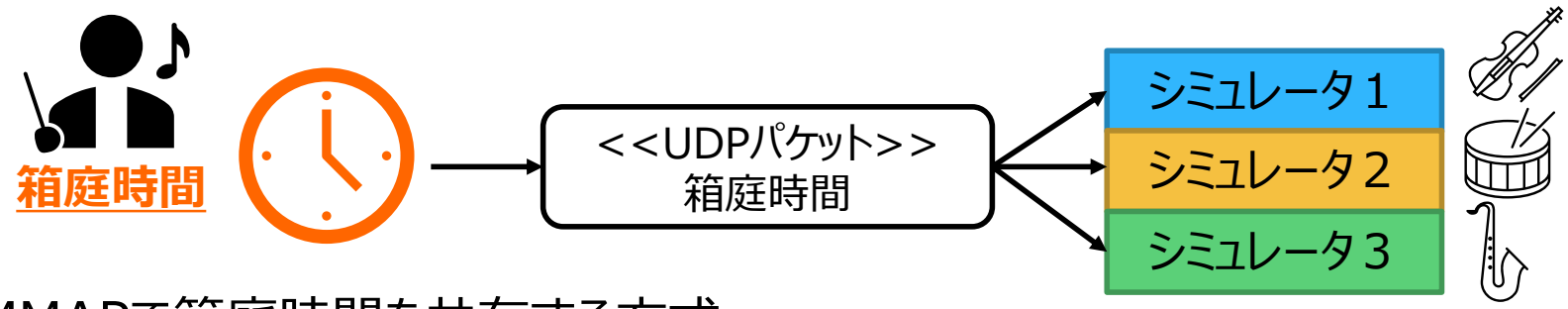
# 箱庭WGで新規に検討した時間同期方式

- **箱庭の時間同期方式**
  - 中央制御方式ではなく，並列化が容易な分散制御方式でのシミュレーション時間同期方式を検討
- **仕組み(ハーモニー)**
  - 各シミュレータは 箱庭時間(マエストロ)をみながら，シミュレーション時間調整し時間同期する(ハーモニー)
    - 箱庭時間より早い場合
      - シミュレーション時間を遅くする
    - 箱庭時間より遅い場合
      - シミュレーション時間を早くする
  - 各シミュレータ時間を可視化
    - 時間同期の程度を定量化
    - 環境スペックの妥当性を評価・調整

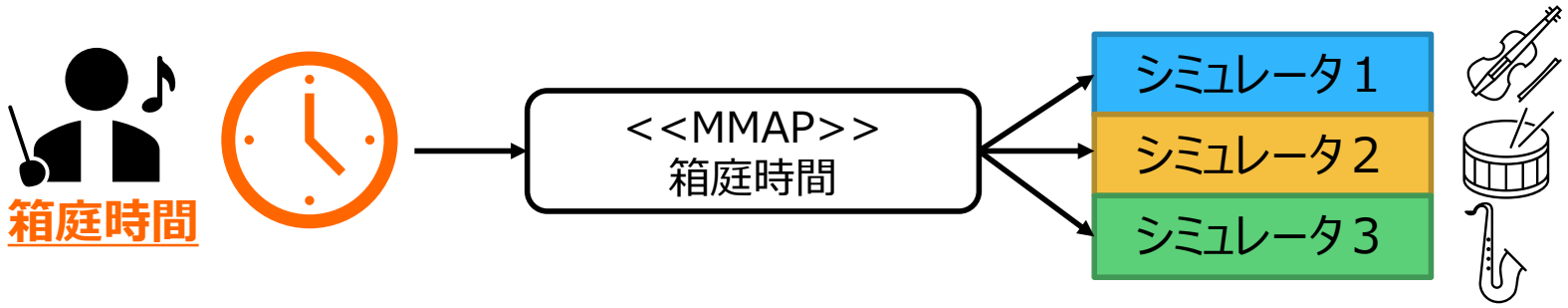


# シミュレータ間の通信方式(UDP/MMAP)

- 箱庭時間の通信方式として以下の２つの方式があります
  - UDPで箱庭時間を共有する方式
    - メリット：各シミュレータは別PCに配置可能であり，負荷分散可能
    - デメリット：ディレイあり




  - MMAPで箱庭時間を共有する方式
    - メリット：ディレイなし
    - デメリット：箱庭時間管理機構と隠しシミュレータは同一PC内に配置する必要あり

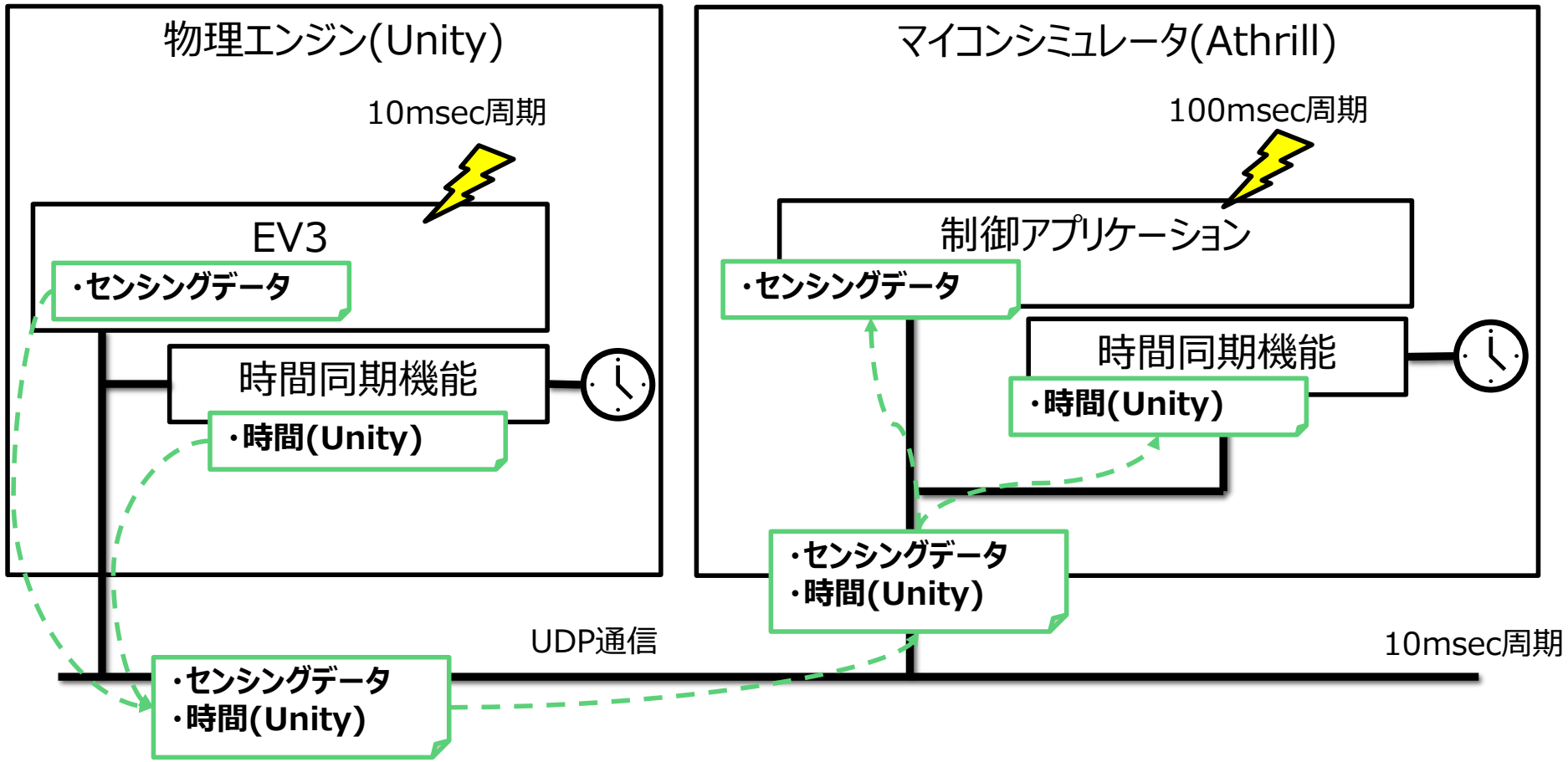


# シミュレータ間通信/時間同期機構

- センサ/アクチュエータ
- シミュレーション時間と時間同期の背景
- シミュレーション時間同期機構
- **ETロボコンシミュレータの通信**

# ETロボコンシミュレータの通信

- UDP方式
  - 物理エンジン(Unity)のセンシングデータ送信タイミングでシミュレーション時間通知する
  - マイコンシミュレータ(athrill)が物理エンジンのシミュレーション時間にあわせて時間調整する



# アジェンダ

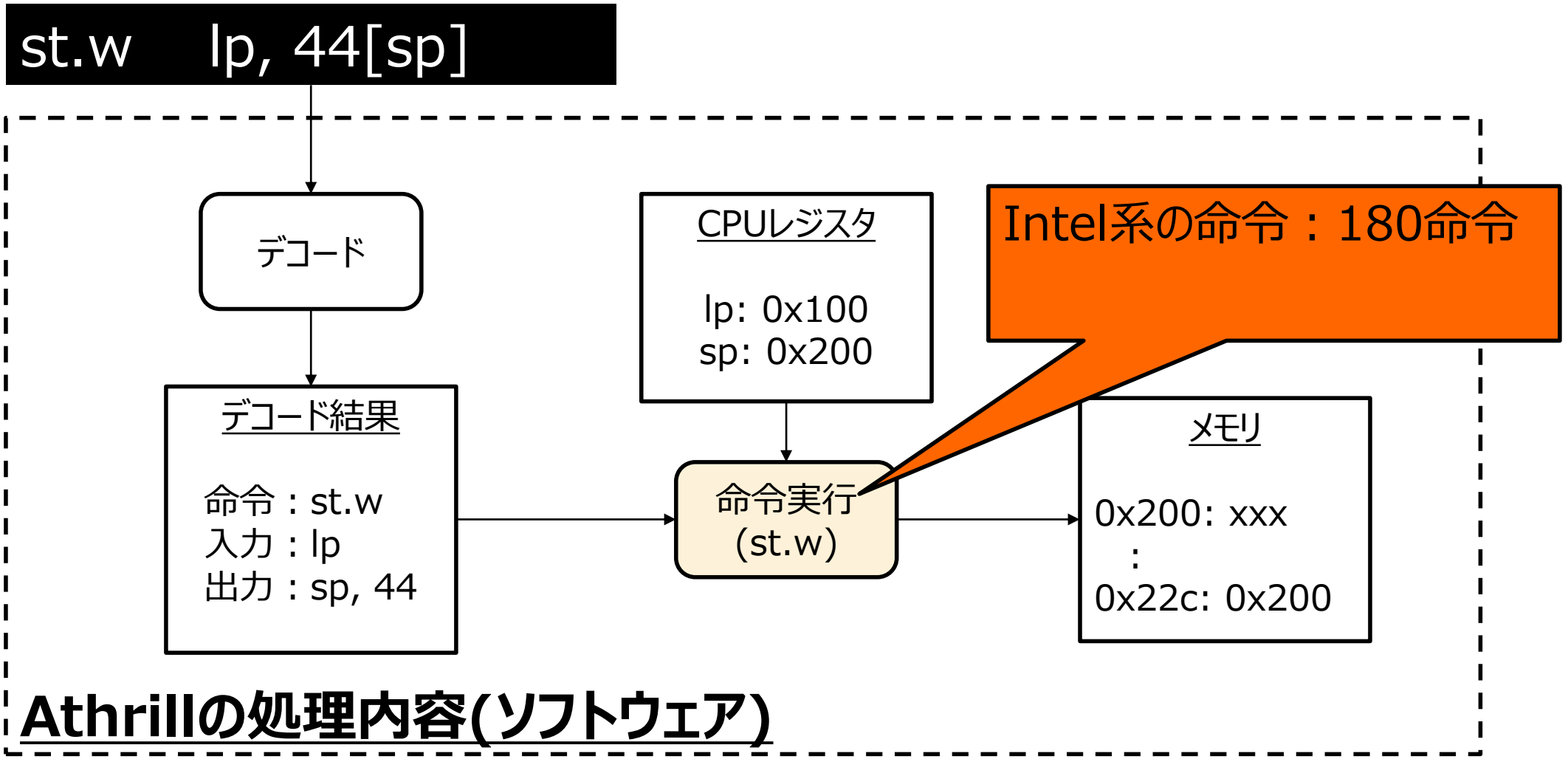
1. 箱庭とは
2. ETロボコンで利用されている箱庭の要素技術
3. 箱庭要素技術とその仕組み
4. **ノウハウ集**
5. 箱庭WGへのお誘い

# ノウハウ集

- マイコンシミュレータが苦手なこと
- マイコンシミュレータ高速化技法あれこれ
- テスト構成あれこれ
- デバッグ技法あれこれ

# マイコンシミュレータが苦手なこと

- CPU命令セットのエミュレーション処理量は意外と多い(命令数：180倍以上)

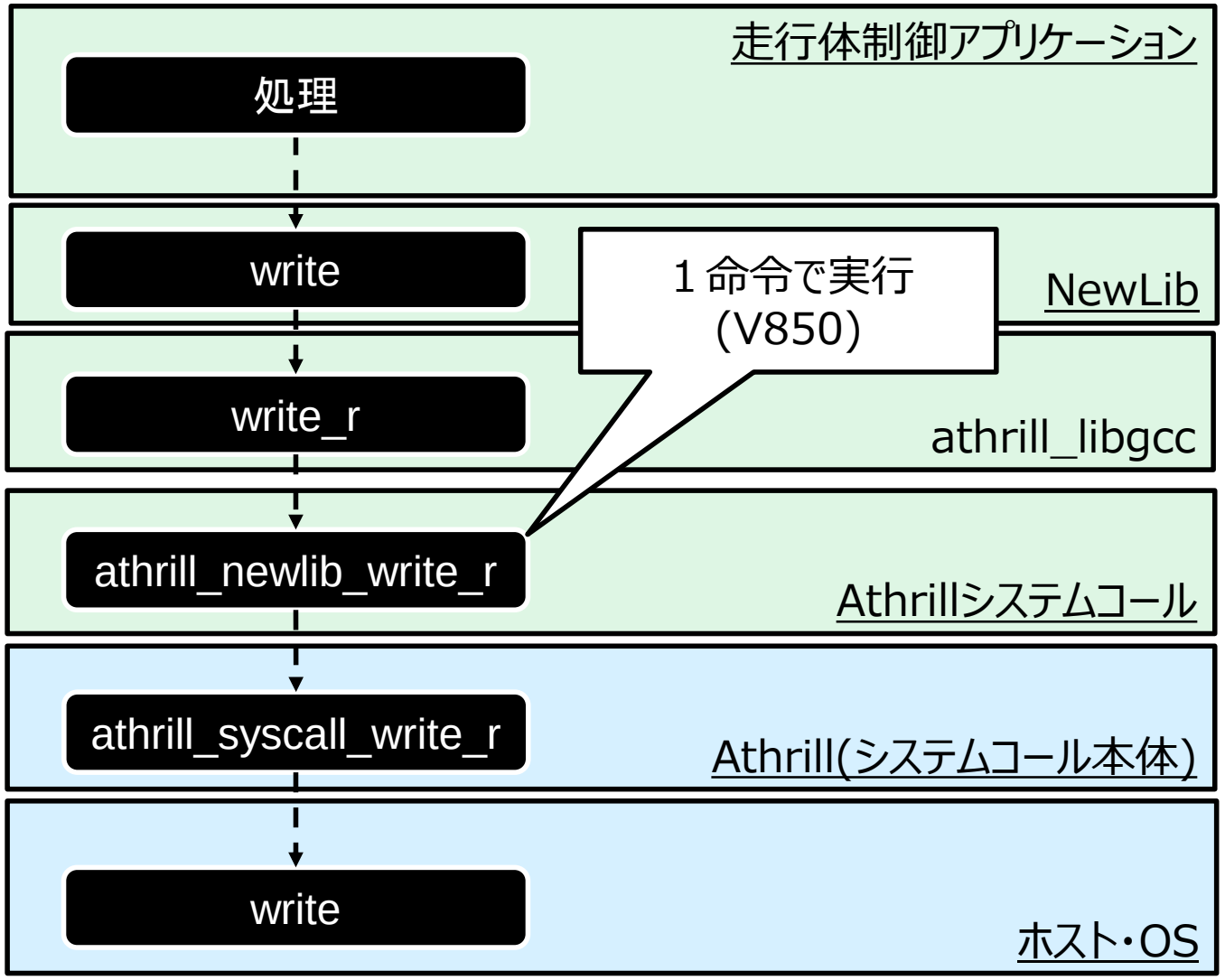


# ノウハウ集

- マイコンシミュレータが苦手なこと
- **マイコンシミュレータ高速化技法あれこれ**
- テスト構成あれこれ
- デバッグ技法あれこれ

# マイコンシミュレータ・高速化技法あれこれ

- ホスト機能呼び出し
  - ホストPCの処理速度で動く
  - ホスト機能をそのまま利用可能




**高速実行**
**(ホストPCの処理速度)**

# マイコンシミュレータ・高速化技法あれこれ

- シミュレーション時間飛ばし機能

| シミュレーション実行対象 | | シミュレーション時間 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| イベント | CPU | 命令実行 | 命令実行 | HALT | HALT | HALT | HALT | 割り込み受付 | 命令実行 | 割り込み受付 | 命令実行 |
| | デバイス1 | – | – | – | – | – | – | 割り込み通知 | – | – | – |
| | デバイス2 | – | – | – | – | – | – | – | – | 割り込み通知 | – |

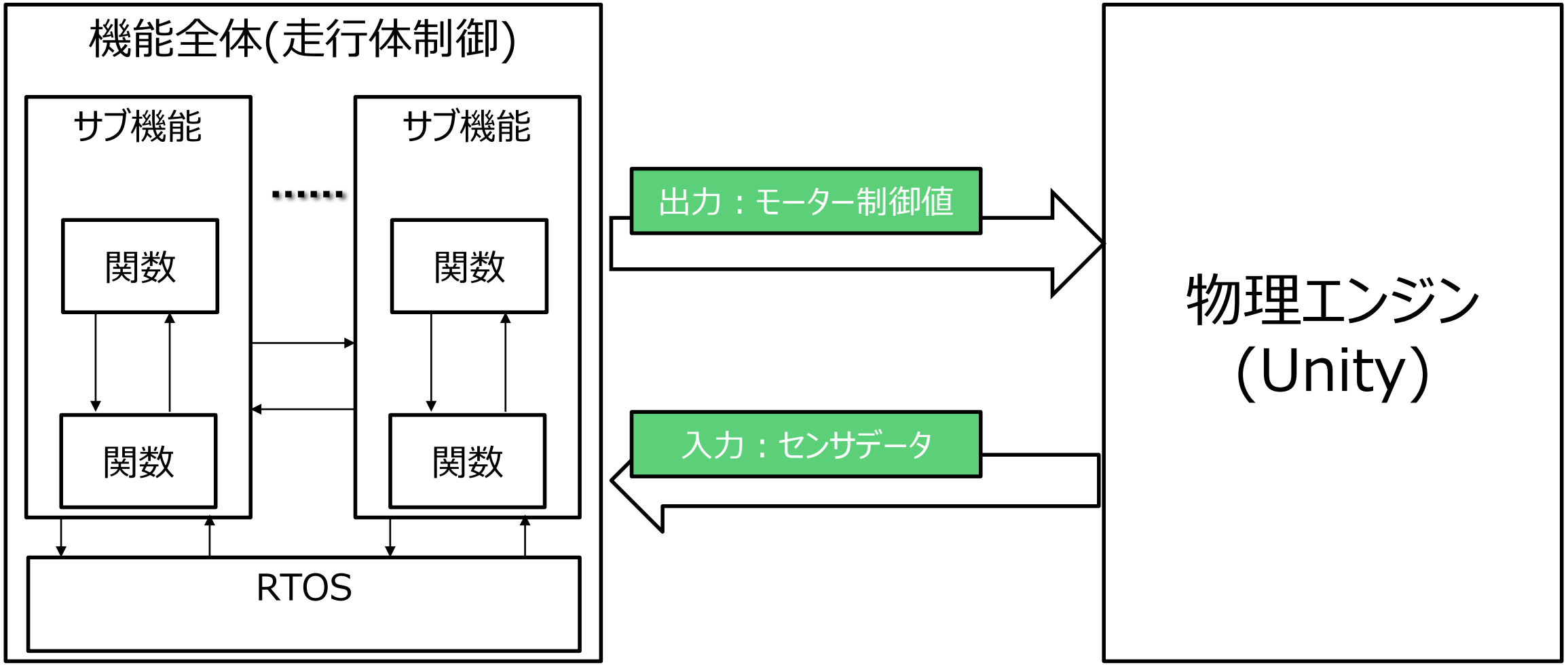
| シミュレーション実行対象 | | シミュレーション時間 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | スキップ時間 | 7 | 8 | 9 | 10 |
| イベント | CPU | 命令実行 | 命令実行 | HALT | – | 割り込み受付 | 命令実行 | 割り込み受付 | 命令実行 |
| | デバイス1 | – | – | – | – | 割り込み通知 | – | – | – |
| | デバイス2 | – | – | – | – | – | – | 割り込み通知 | – |

# ノウハウ集

- マイコンシミュレータが苦手なこと
- マイコンシミュレータ高速化技法あれこれ
- **テスト構成あれこれ**
- デバッグ技法あれこれ

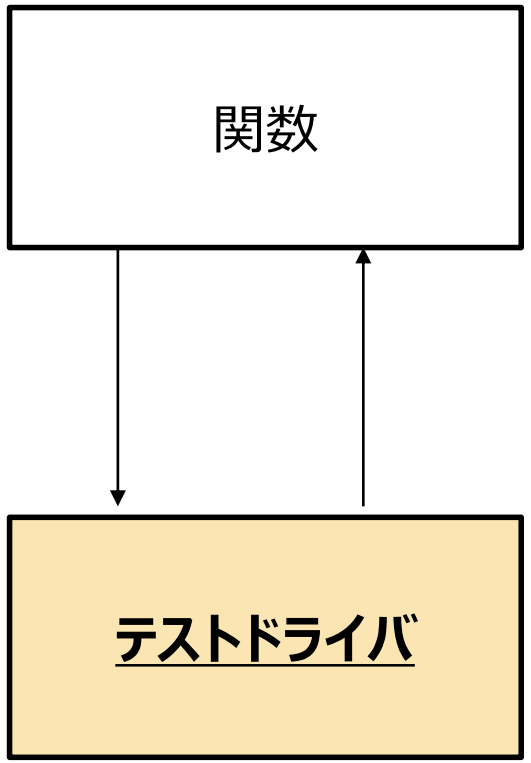
# テスト構成あれこれ

- テストで確認するレベルは色々あります(関数，サブ機能，機能全体)



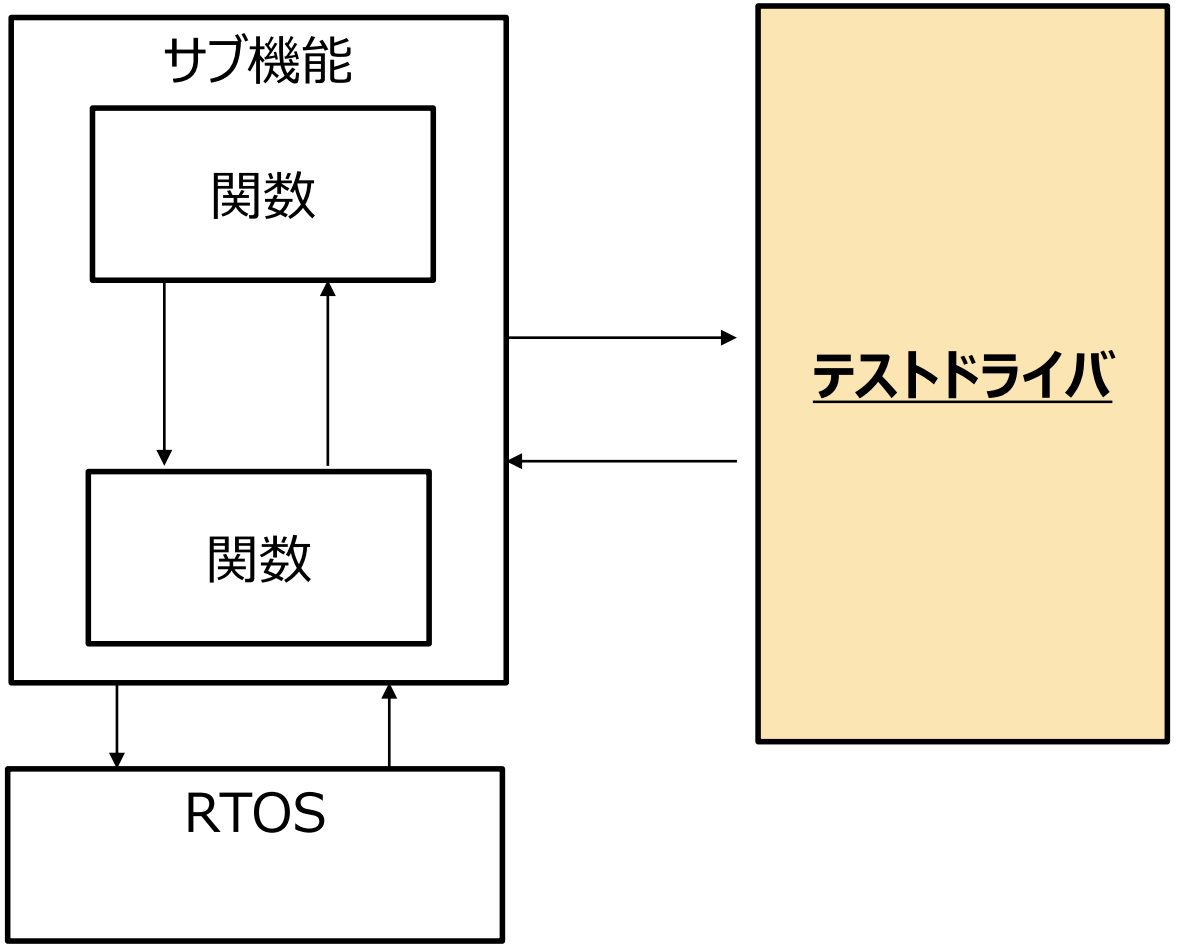
# 関数毎の単体テスト

- 以下のフリーのC言語単体テストフレームワークは，Athrill上でも実行できます．
  - **Unity(テストフレームワークの方)**
    - https://github.com/ThrowTheSwitch/Unity
    - サンプルは，以下で公開しています．
      - https://github.com/tmori/unit-test-sample
  - **gmock, gtest**
    - http://opencv.jp/googlemockdocs/fordummies.html
    - v850向けのクロスコンパイラを利用すれば動作するはず…
- コードカバレッジツール gcov とも連携できます！



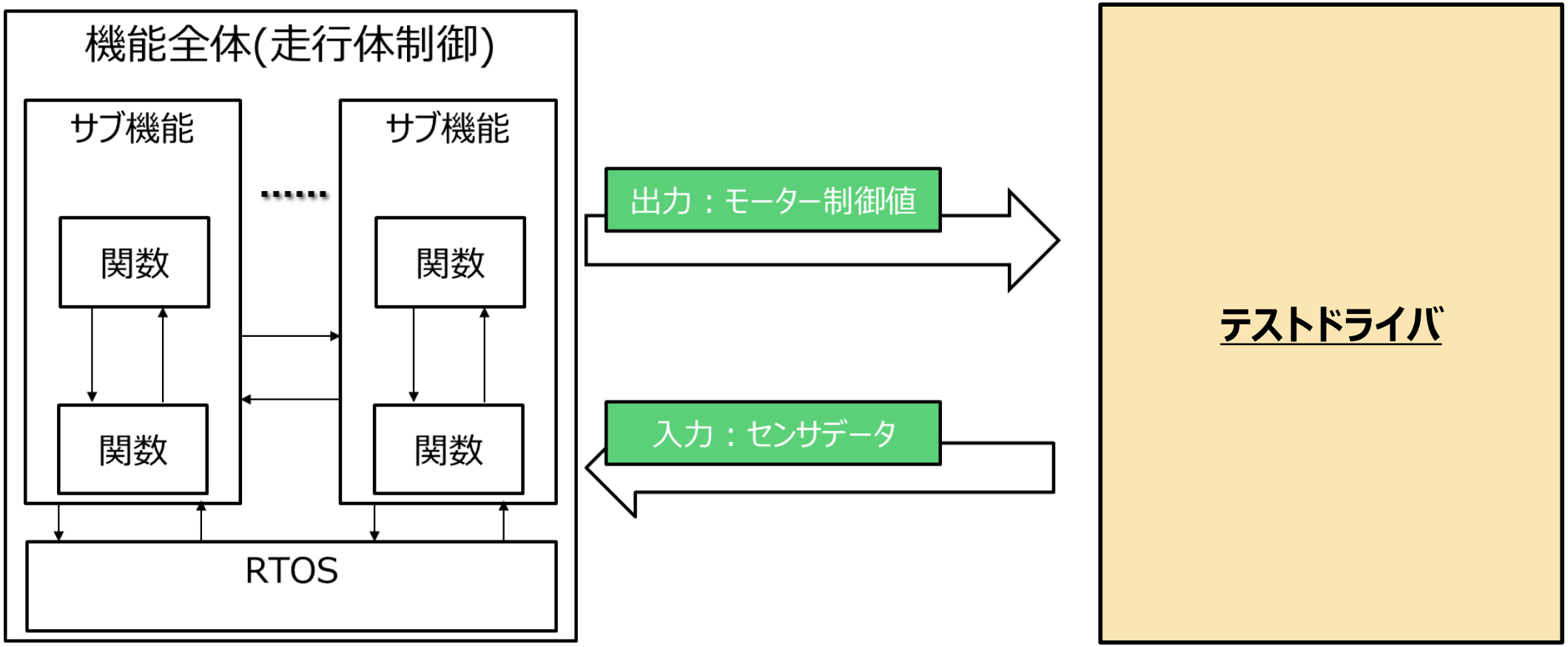
# サブ機能毎の単体テスト

- RTOSを結合した状態でのテスト
  - AthrillはRTOSをシミュレーション可能
  - タスク単位でのテストも可能



# 機能全体の単体テスト

- 機能全体の単体テストのための構成として，Unityスタブ(テストドライバ)を用意して，機能全体のテストを行うことも可能です．



# ノウハウ集

- マイコンシミュレータが苦手なこと
- マイコンシミュレータ高速化技法あれこれ
- テスト構成あれこれ
- **デバッグ技法あれこれ**

# デバッグ技法あれこれ

- Athrillのデバッガ
- Athrillのデバッグ機能を使う
- 内部デバッグ情報を可視化する

# Athrillのデバッガ

- Athrillデバッグモードで起動すると，デバッガを利用できるようになります
- デバッグ用のコマンド入力待ちになります

```
$ make debug
```

```
athrill2 -c1 -i -m ../common/memory.txt -d ../common/device_config.txt -t -1 asp
core id num=1
ROM : START=0x0 SIZE=2048
RAM : START=0x200000 SIZE=2048
RAM : START=0x5ff7000 SIZE=10240
RAM : START=0x7ff7000 SIZE=10240
 :
DEBUG_FUNC_VDEV_TX_IPADDR = 127.0.0.1
VDEV:TX IPADDR=127.0.0.1
VDEV:TX PORTNO=54001
VDEV:RX IPADDR=127.0.0.1
VDEV:RX PORTNO=54002
 :
[DBG>
HIT break:0x0
```

# Athrillのデバッグ機能を使う

- 動作モード切替
  - c コマンド
  - q コマンド

```
[DBG>
HIT break:0x0
c
[CPU>brick_dri initialized.

  ______  _______ ___  _______
 / __/ | / /_  // _ ¥/_  __/
/ _/ | |/ //_ </ , _/ / /
/___/ |___/____/_/|_| /_/

====================>Beta 7<=

Powered by TOPPERS/ASP3 RTOS of Hakoniwa
Initialization is completed..
#### motor control start
```

```
q
[NEXT> pc=0x5796 prc_support.S 388
[DBG>
```

# Athrillデバッグ機能あれこれ

- https://qiita.com/kanetugu2018/items/cf3dea16710a3f0737e8#%E3%83%9E%E3%83%8B%E3%83%A5%E3%82%A2%E3%83%AB

  - CPU実行ログ表示切り替え(v)
  - ブレーク設定(b)
  - ステップ実行(n)
  - 関数実行トレース表示(ft)
  - CPUレジスタ表示(cpu)
  - メモリ情報表示(p)
  - データウォッチ(w)
  - 消費クロック(e)
  - 関数プロファイル(prof)

# ちょうど先週mrubyの動作確認で調査依頼がありました

[現象]

シミュレーション実行すると AthrillがROM領域書き込みを検出して停止

```
 :
mpu_put_data8:error: can not write data on ROM :addr=0x47a5f data=2
Exec Error code[0]=0x63ff code[1]=0x15c7 type_id=0x0 code_id=167
CPU(pc=0x6023) Exception!!
```

[原因]

スタックオーバーフローが発生していました

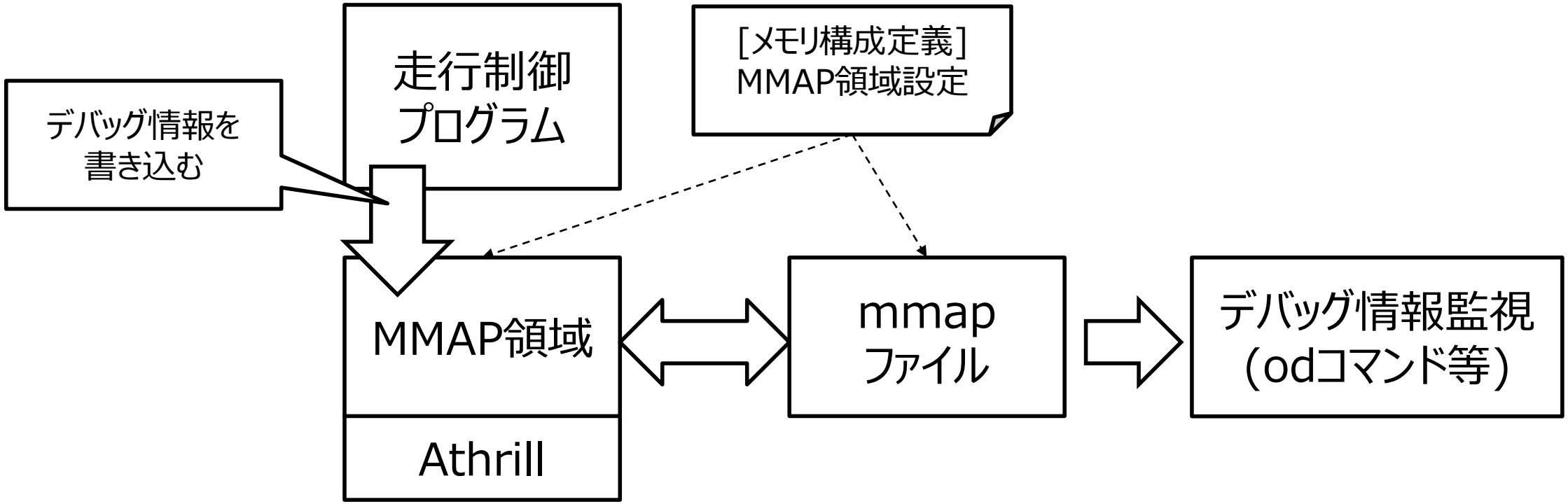
[調査方法]

- 関数実行トレース表示
- CPUレジスタ表示
- CPU実行ログ表示
- objdump コマンド(問題発生個所のアセンブラ確認)
- readelf コマンド(変数シンボルのアドレス/サイズ確認)

[可能なら]
メモリ系の不具合解析時には，アセンブラレベルの解析が求められますので，v850のアセンブリ言語/CPUレジスタあたりを知っていると良いです～.

# 内部デバッグ情報を可視化する

- AthrillのMMAP機能を使って内部デバッグ情報の可視化
  1. Athrillのメモリ構成定義ファイル(memory.txt)にMMAPファイル設定を追加
  2. memory.txtで参照しているファイル実体を作成する
  3. 走行制御プログラムから，memory.txtで定義したMMAP領域にデバッグ情報を書き込む



# 内部状態可視化のデバッグ例

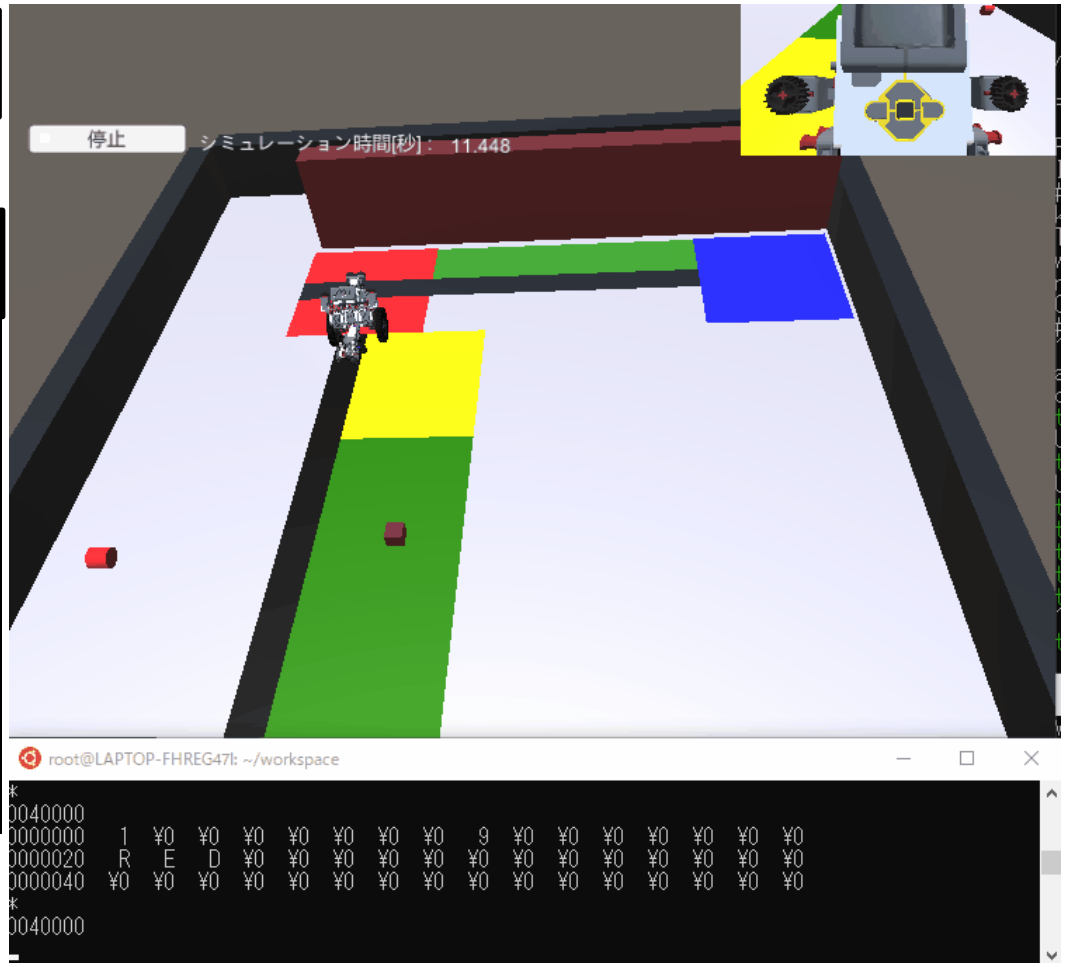
## メモリ構成定義ファイル

```
MMAP, 0x20000000, /root/workspace/sdk/workspace/base_practice_1/debug.bin
```

## 走行制御プログラム

```
typedef struct {
        char phase[8];
        char ultrasonic[8];
        char color[16];
} DebugInfoType;
static DebugInfoType *debug_infop = ((volatile DebugInfoType*)0x20000000);

memset(debug_infop, 0, sizeof(DebugInfoType));
snprintf(debug_infop->ultrasonic, 8, "%d", ultrasonic_value);
snprintf(debug_infop->phase, 8, "%d", Practice2_Phase);
memcpy(debug_infop->color, color_str, strlen(color_str));
```



# おわりに

- **でっかく語って，少しずつ育てております！**
  - https://toppers.github.io/hakoniwa/
- **箱庭WGの狙い・趣旨にご賛同いただける方の参画をお待ちしております！！**
  - まずはSlackでの議論，活動内容へのご要望，コア技術や各アセットの開発，などに参加したい方
  - 箱庭WGの活動で期待される技術成果を活用したい方

**よろしくお願いいたします！！**