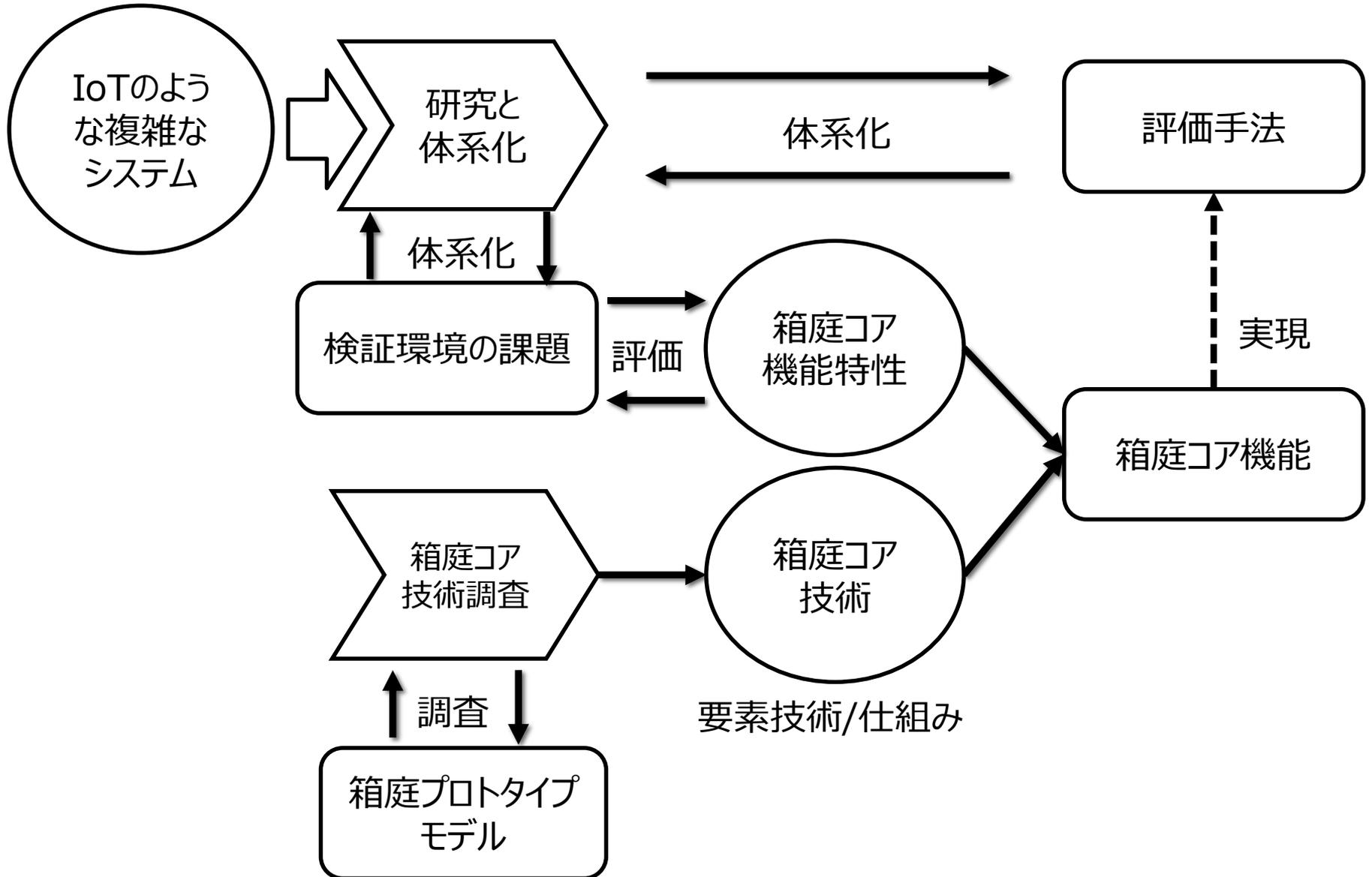


箱庭コア技術紹介

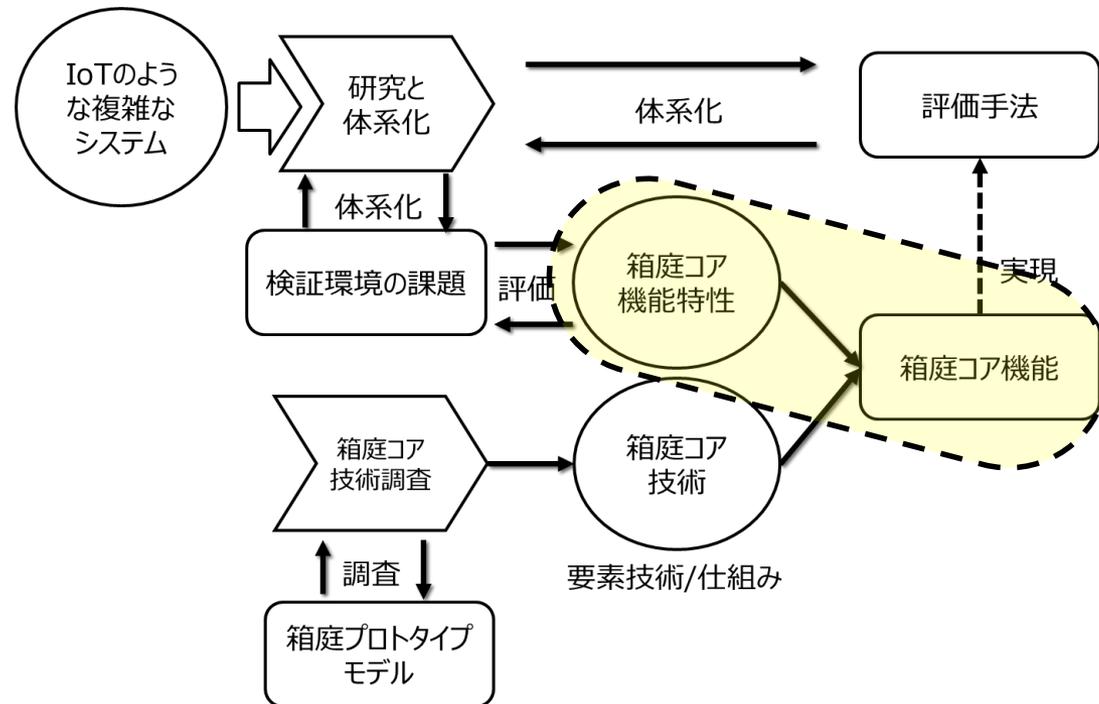
- 箱庭コア機能検討の全体像
- 箱庭コア機能の紹介
- IoTのような複雑なシステムの検証に関する研究と体系化
- 箱庭プロトタイプモデル
- 箱庭コア技術の検討状況

箱庭コア機能検討の全体像



箱庭コア機能の紹介

- 箱庭の基本コンセプト
- 箱庭のアーキテクチャ
- 箱庭コア機能紹介
- FMI (Functional Mock-up Interface)

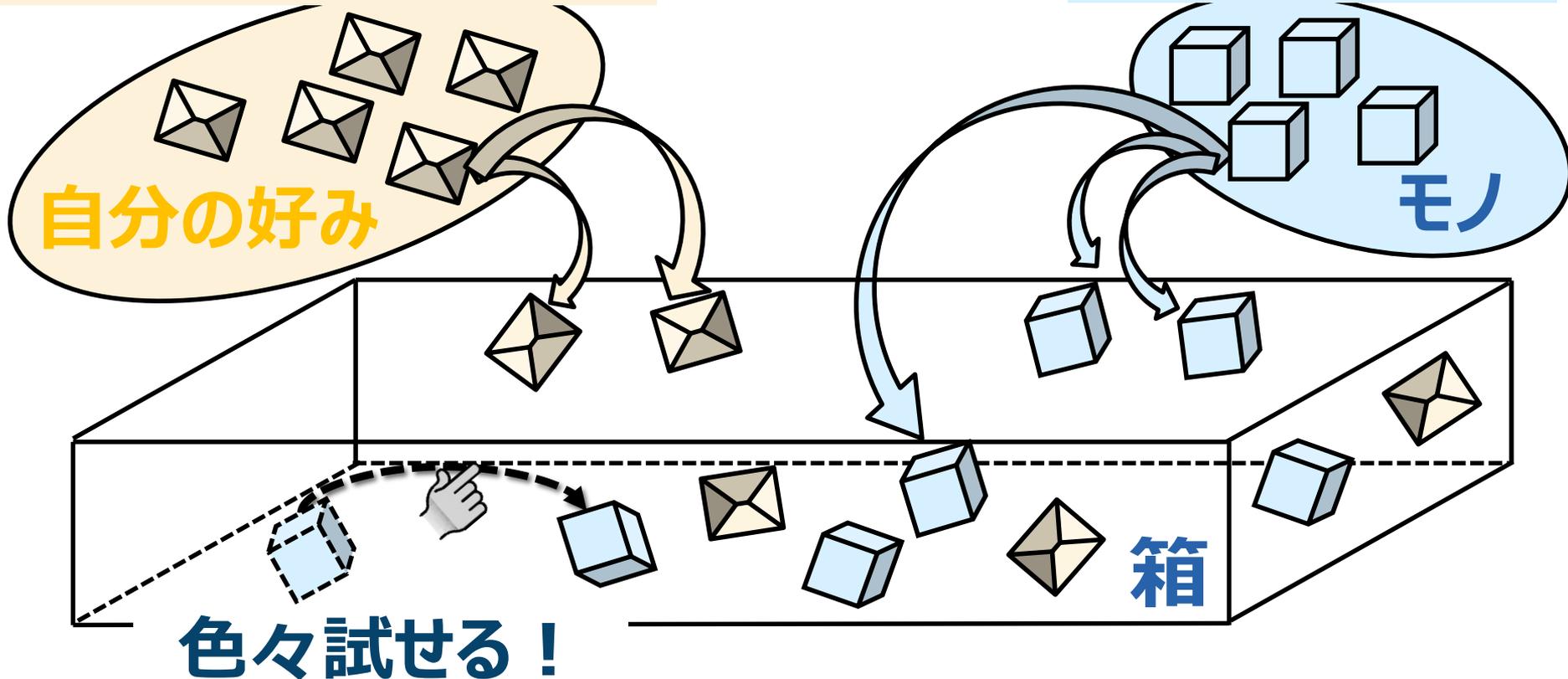


箱庭の基本コンセプト

- 箱の中に,
- いろいろなモノを自分の好みに配置して
- 色々試せる!

評価シナリオ, 自社サービスP/F, 地図等

車, 環境, イベント等々

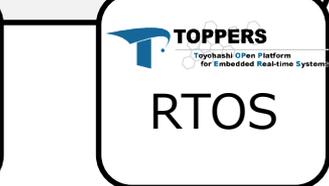
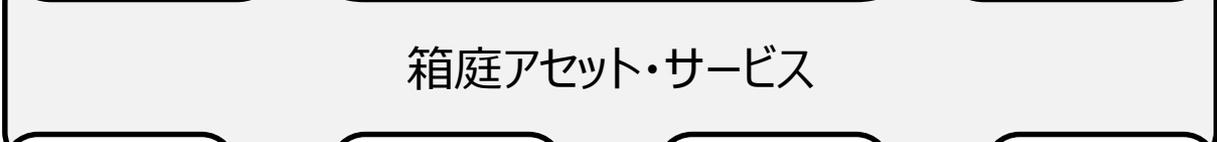
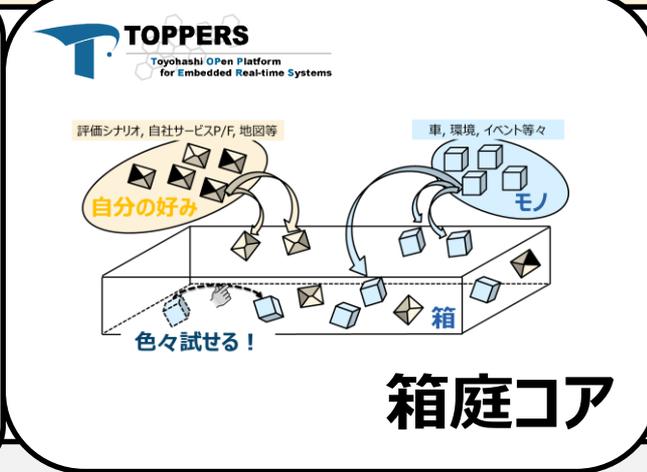
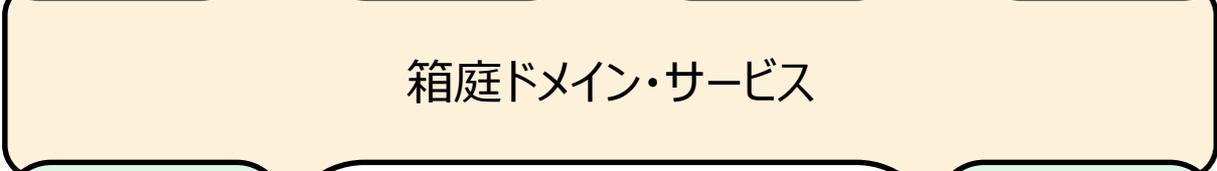
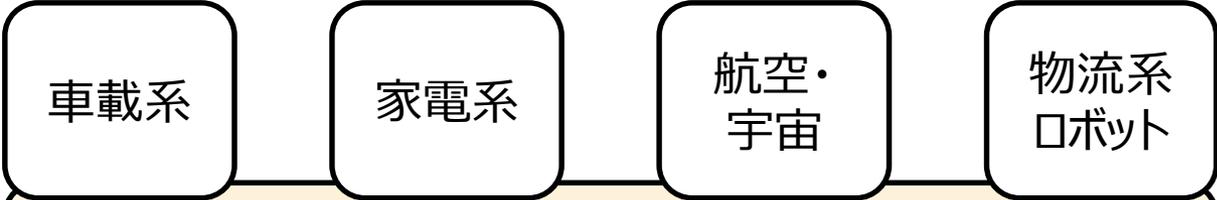


箱庭のアーキテクチャ

■ 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



箱庭コア機能紹介

箱庭の核となる機能(カーネル)は以下の4種類

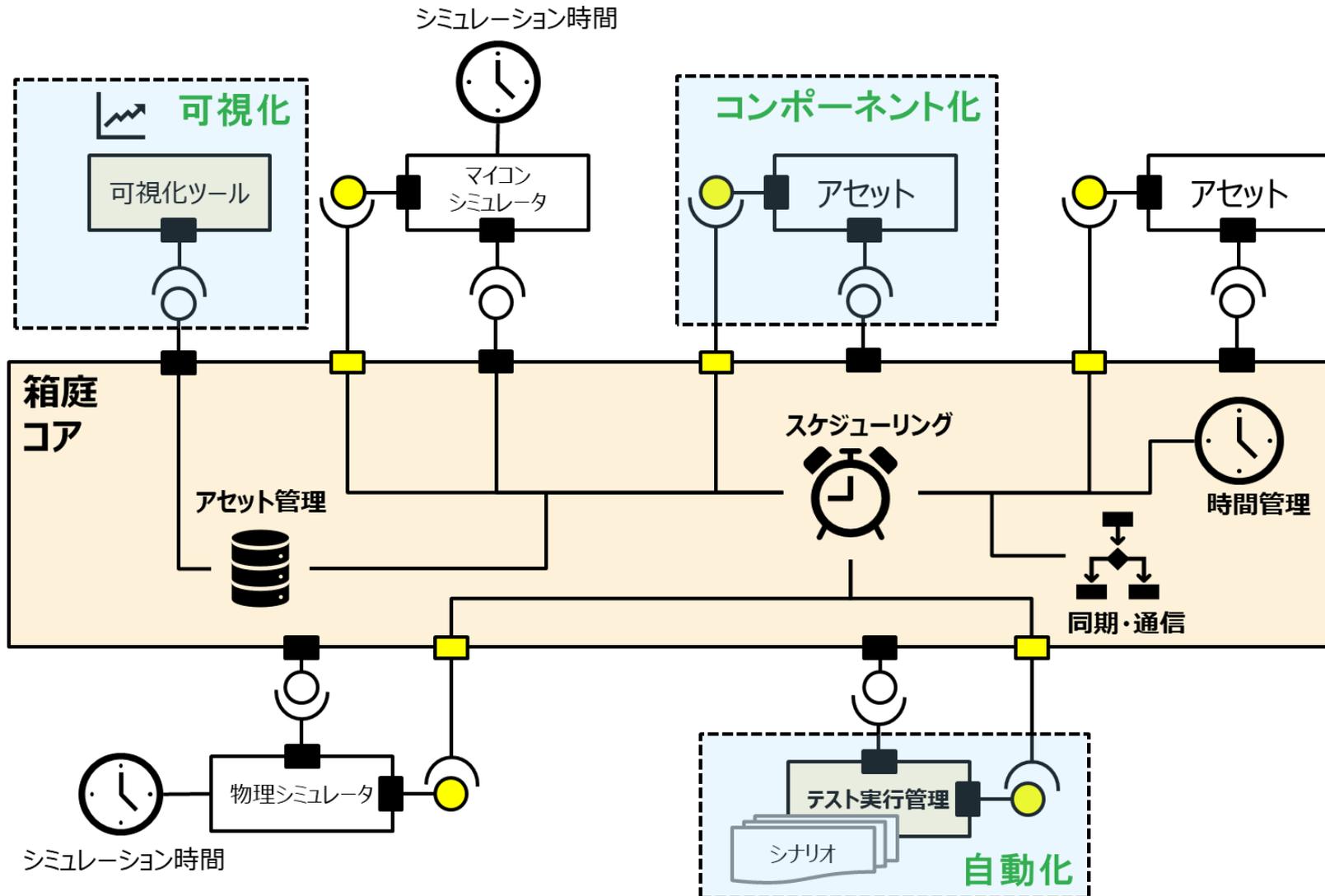
1. スケジューリング
2. 同期・通信
3. 時間管理
4. アセット管理

機能の切り方はWG内でも意見が
分かれているので、一旦、仮置きです

箱庭コア機能が備えるべき重要な機能特性

1. コンポーネント化
2. 可視化
3. イベント駆動化
4. 自動化

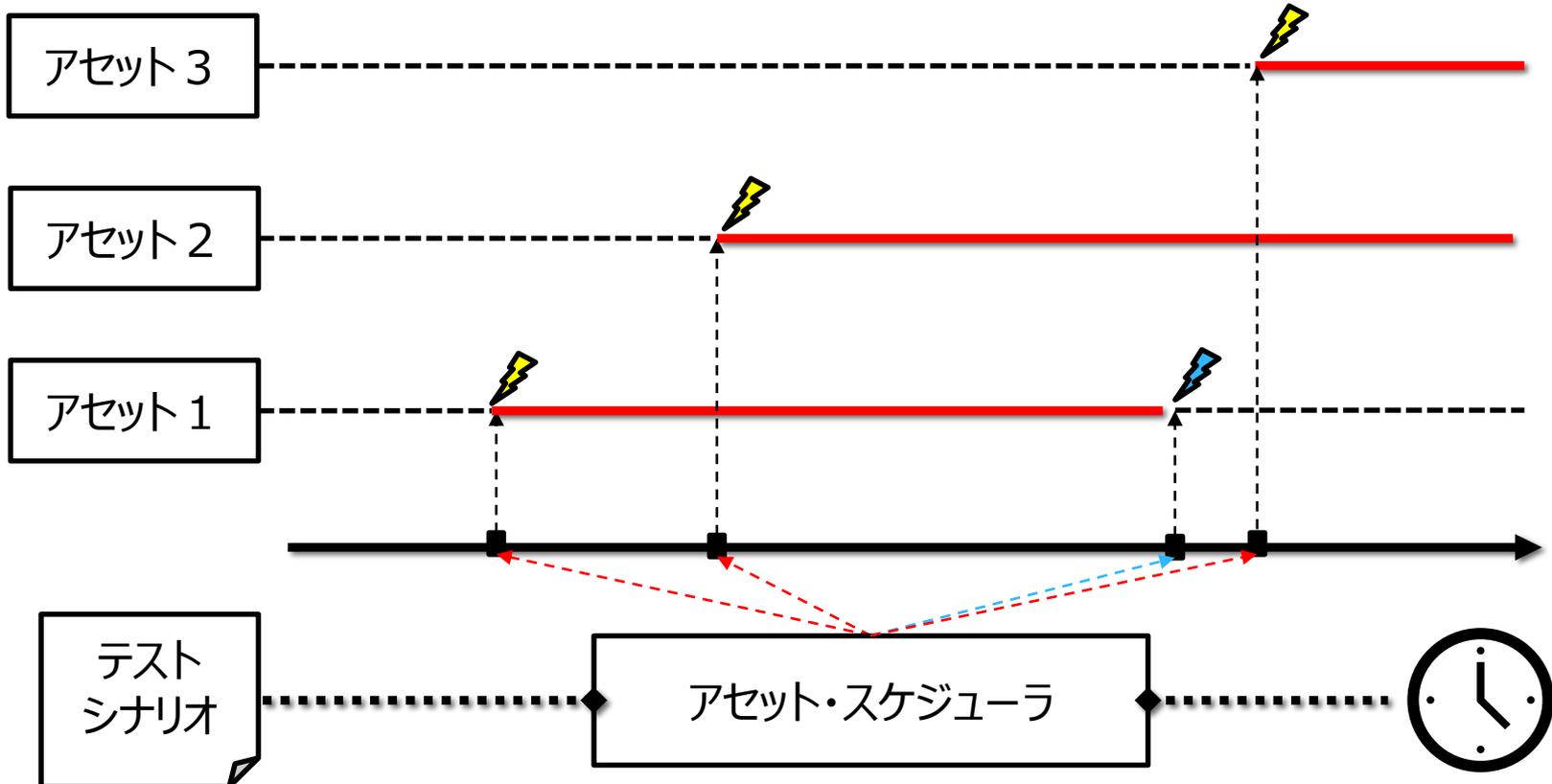
箱庭コア機能と機能特性



スケジューリング

- 箱庭にプラグインされたアセットの実行タイミングを管理する機能
- ※機能詳細検討はこれから

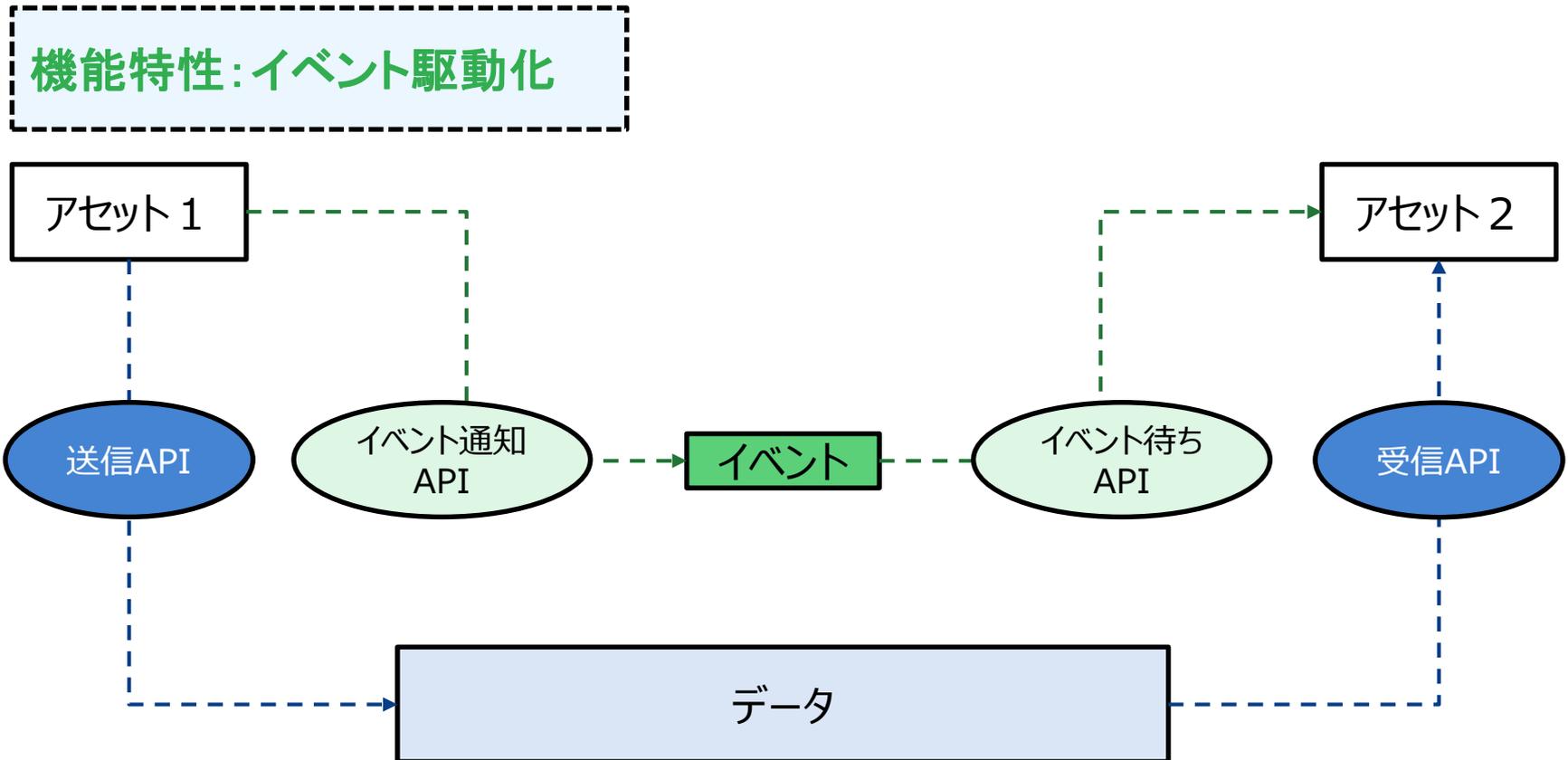
機能特性: イベント駆動化, 自動化



同期・通信

- 箱庭にプラグインされたアセット間でデータ交換するための仕組み
- アセット間でイベント通知/待ちするための仕組み

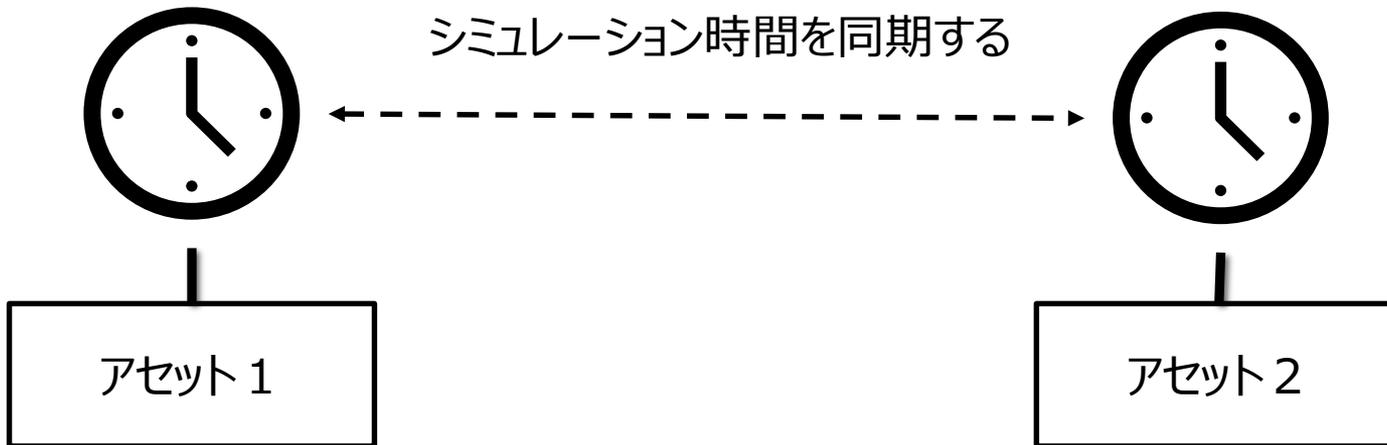
※機能詳細検討はこれから



時間管理

- 箱庭にプラグインされたアセットのシミュレーション時間を管理する機能
 - ※シミュレーション時間同期方式の技術検討が最重要課題

★FMI仕様との互換性を視野に入れる



アセット管理

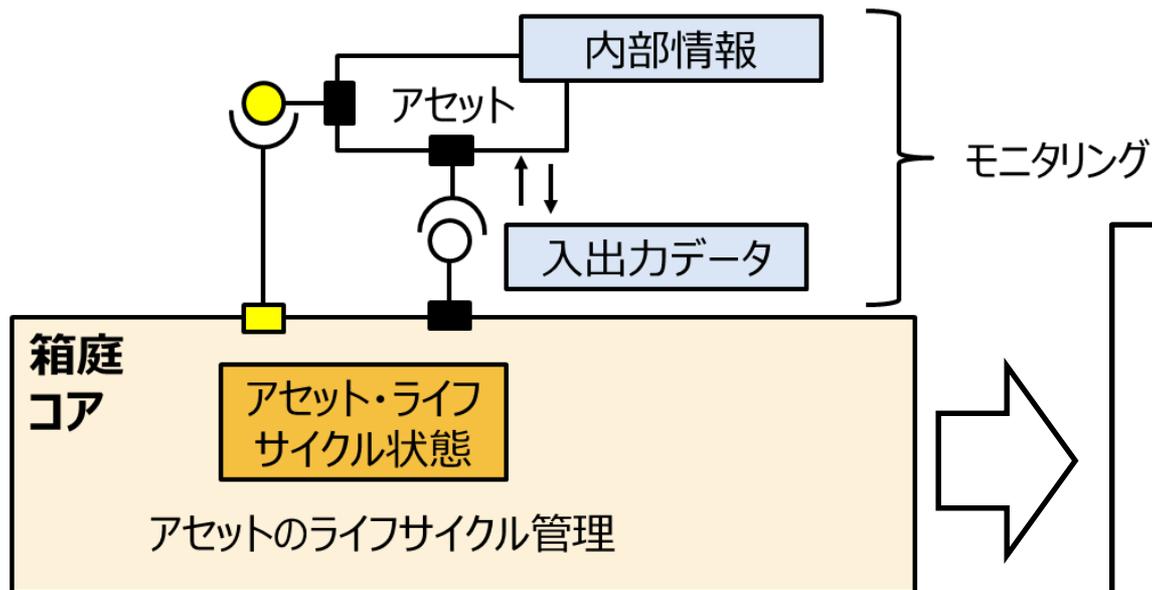
- 箱庭にプラグインされたアセットを管理する

1. アセットのライフサイクル管理(登録~起動~停止~削除)
2. 出力コントロール機能(故障注入等)
3. アセットのモニタリング機能
 - 内部情報(状態, 位置, 速度, 角度等)
 - 入出力データ

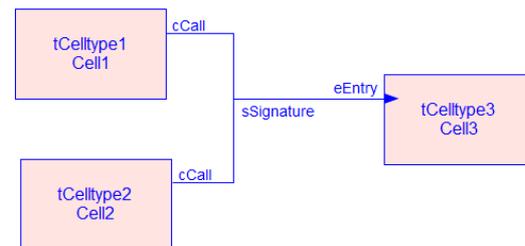
※機能詳細検討はこれから

機能特性:

- コンポーネント化
- イベント駆動化
- 可視化
- ★FMIとの互換性も視野に入れる

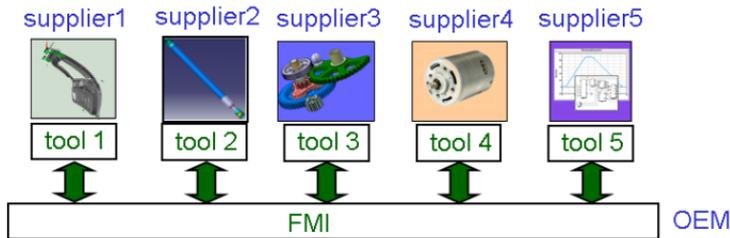


TOPPERS TECSとの連携

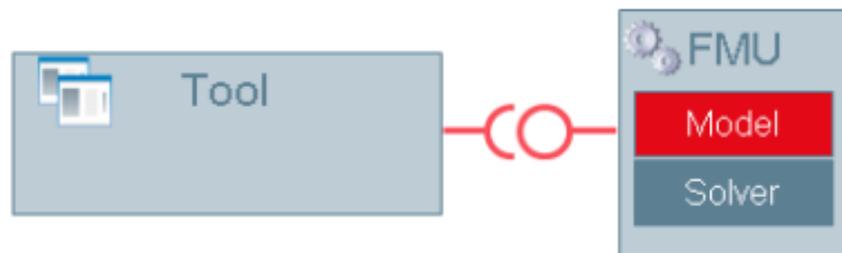


FMI (Functional Mock-up Interface)

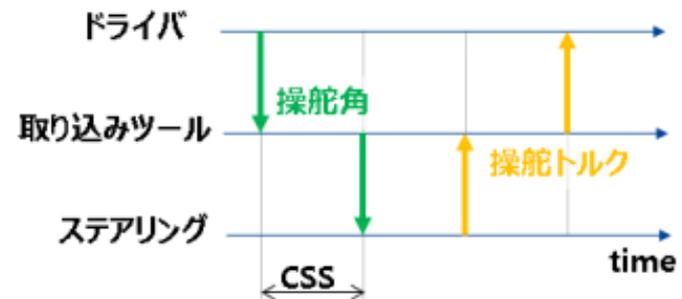
- 欧州の公的プロジェクトが規格化
- シミュレーションツールに依存しないモデル接続のための共通インターフェース



- FMUという単位でToolと接続/シミュレーションする
- シミュレーション時間についても言及されている(調査中)



箱庭アセット管理とほぼ同じ考え方



シミュレーション時間について言及

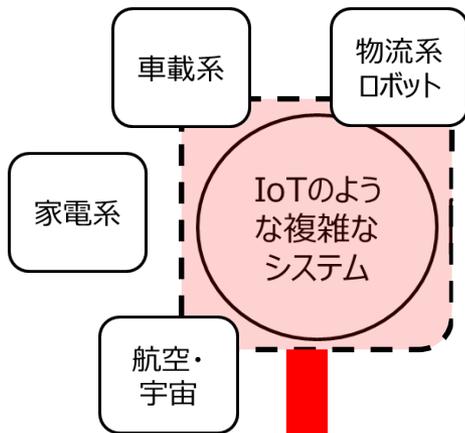
※図は「自動車技術会 自動車制御とモデル部門委員会 FMI 活用・展開検討 WG」の資料を参照
 ※ https://www.jsae.or.jp/katsudou/docu/1035/fmi_guide101.pdf

箱庭プロトタイプモデル毎の 箱庭コア機能の技術検討注力ポイント

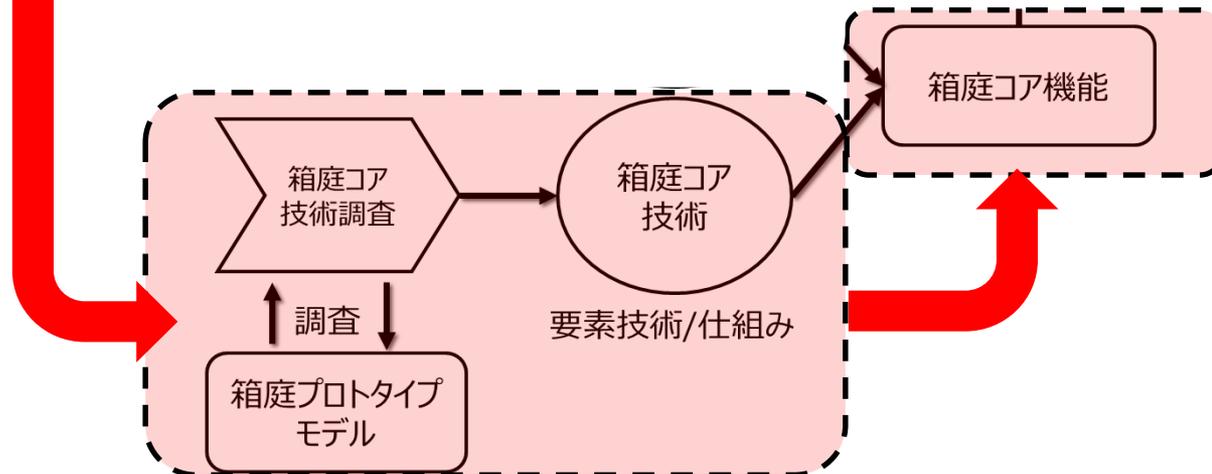
箱庭コア機能	箱庭プロトタイプモデル		
	ETロボコン シミュレータ	ROS・マルチECU向け シミュレータ	車車間協調動作向け シミュレータ
時間管理	○	○	○
同期・通信	—	○	○
スケジューリング	—	○	○
アセット管理	—	—	○

背景

- 箱庭WG結成時は、下の流れで箱庭コア技術を検討しようとしていましたが、いくつか課題があると気づきました。

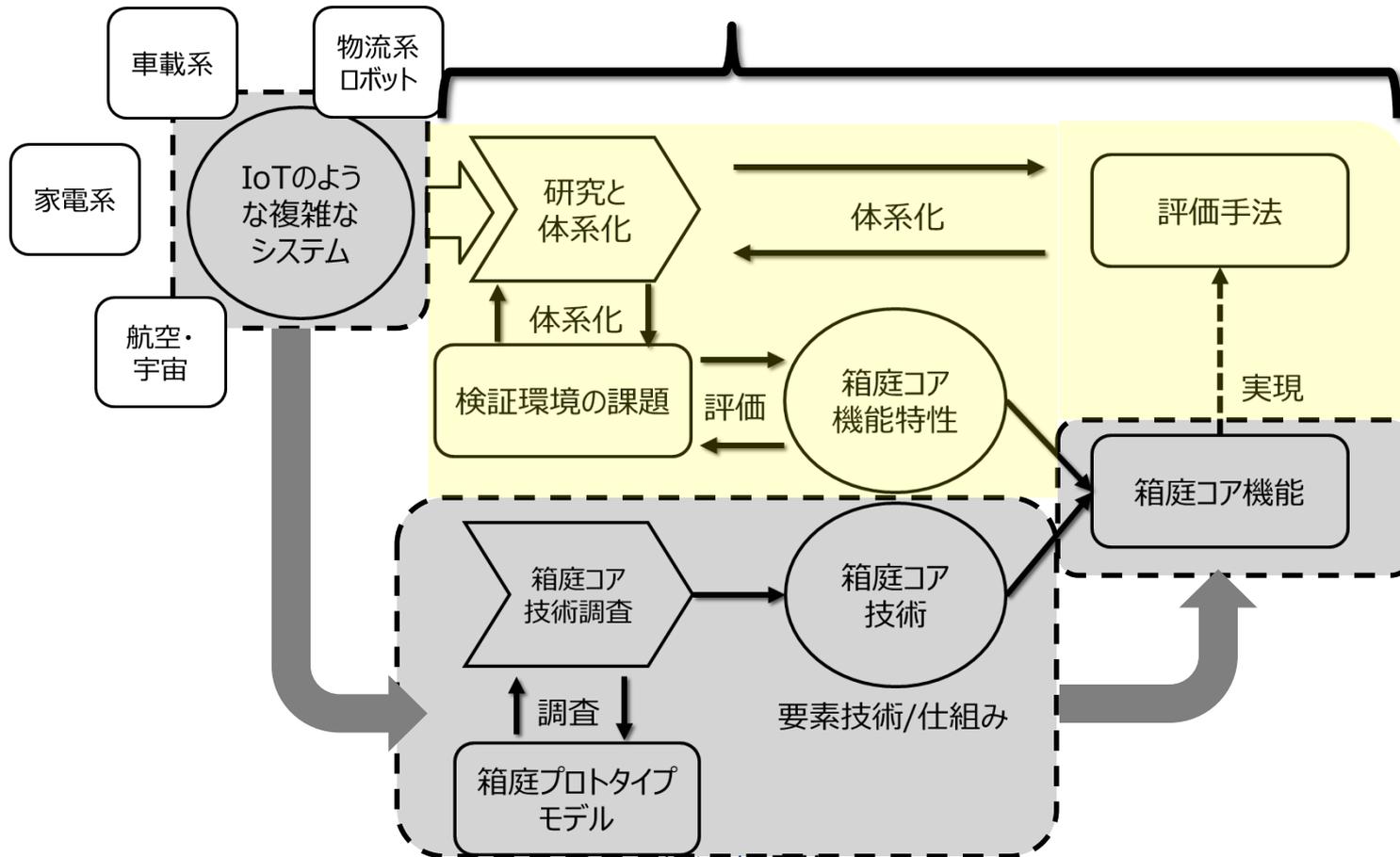


- 複雑系の検証環境として箱庭は汎化されているのか？
- 様々な分野の検証課題を解決できているのか？
- 箱庭を使うと、「何が」、「どうやって」検証できるようになるのか？
⇒ある程度、客観的な指標等がないと説明できない

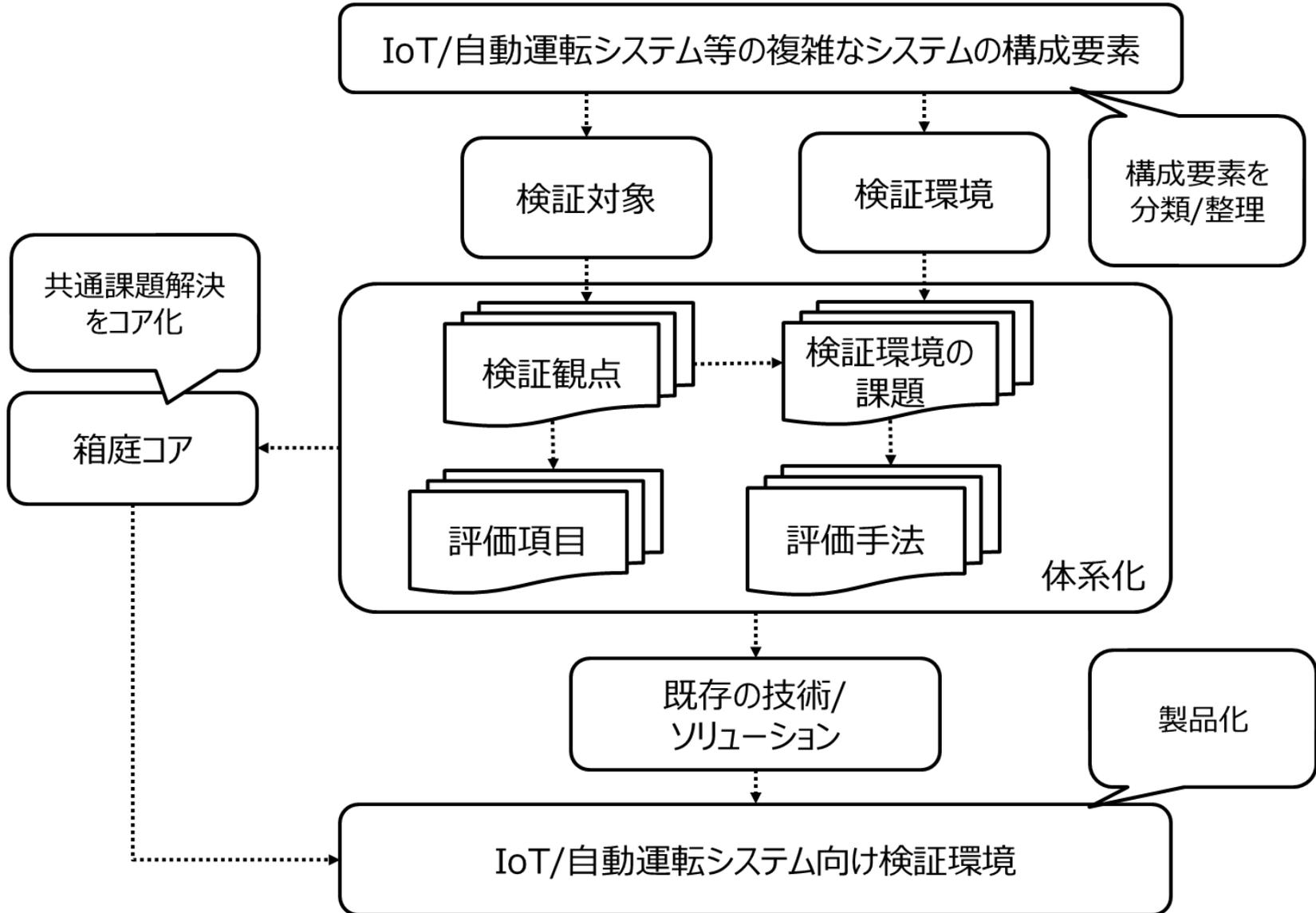


背景

- 客観性を持たせるには，IoTのような複雑なシステムの検証課題や評価手法を整理した上で，箱庭コア機能の妥当性を説明できるようにすべきではないか？

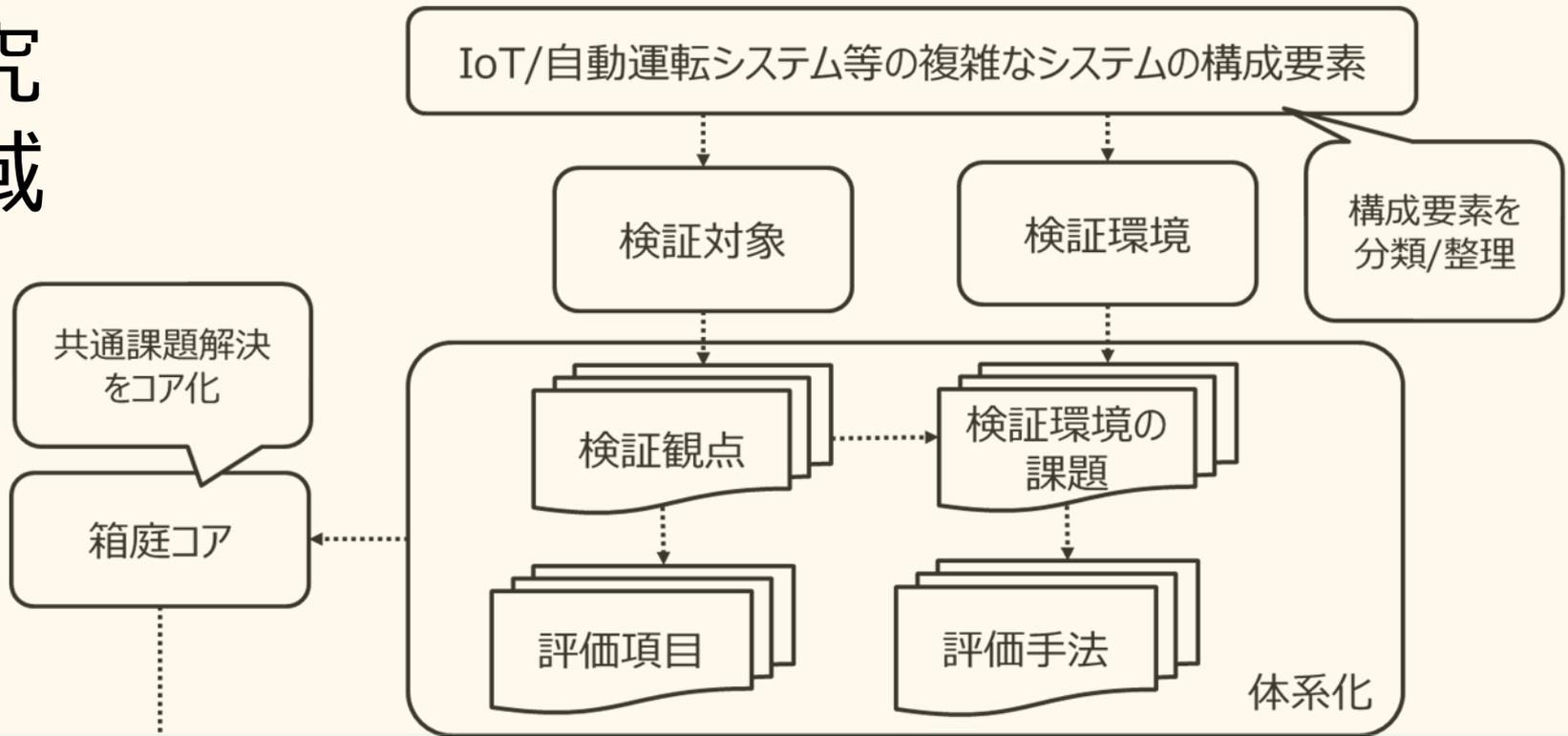


進め方

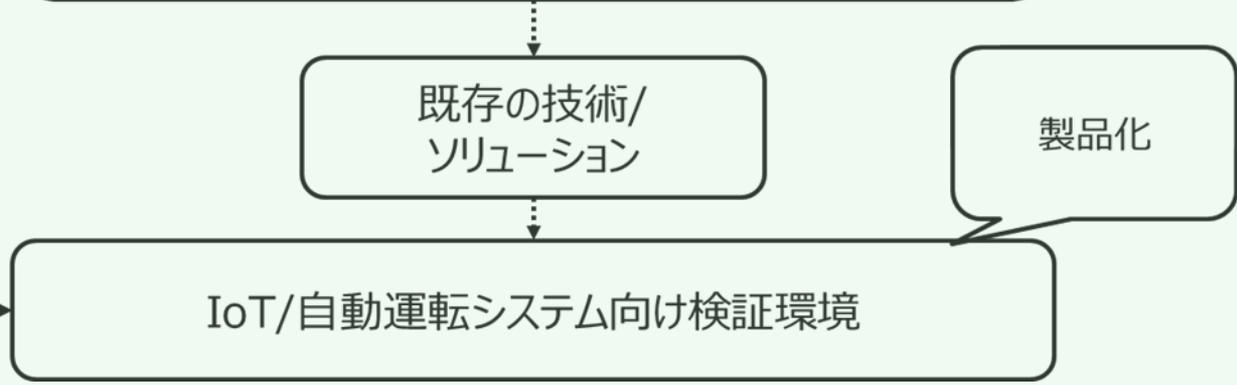


進め方

研究領域

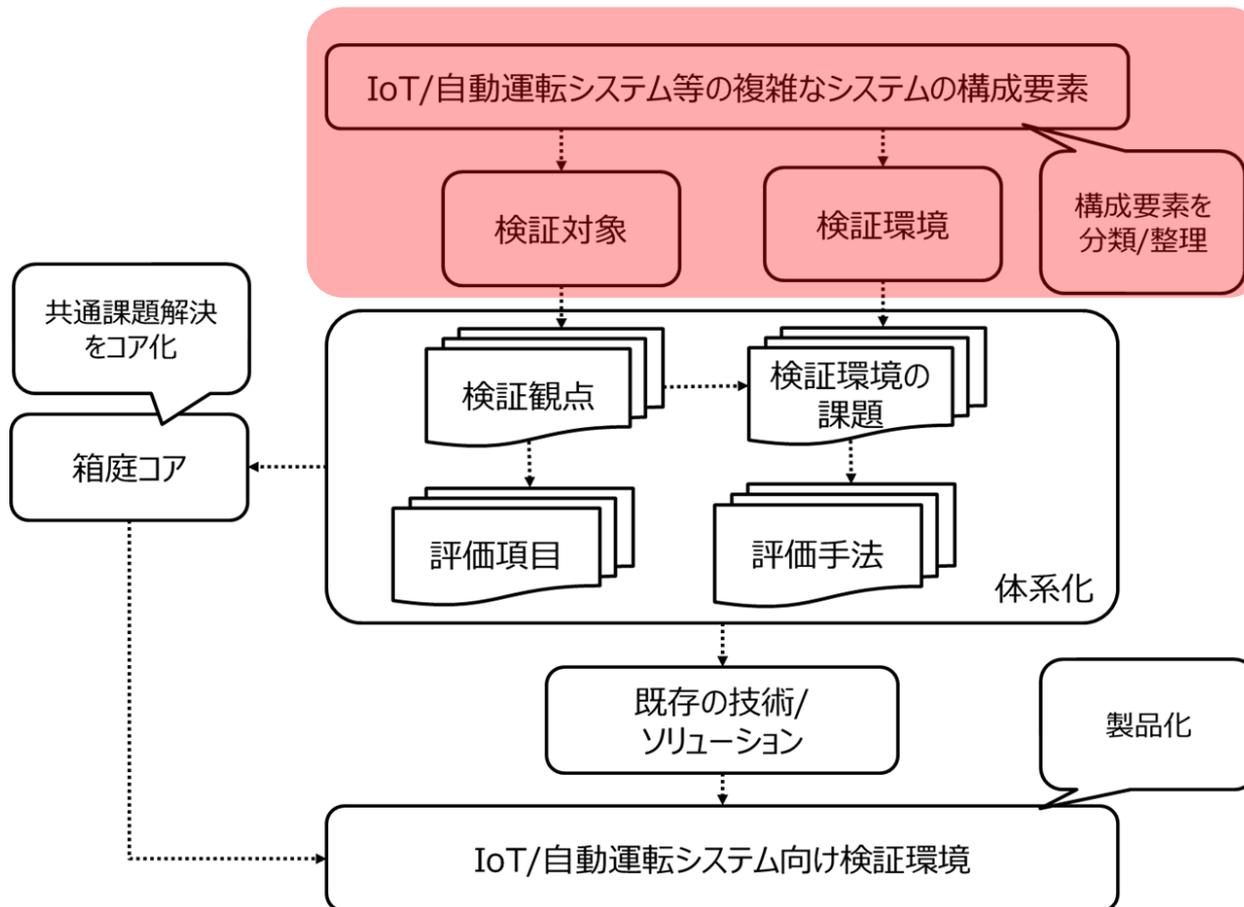


ビジネス領域



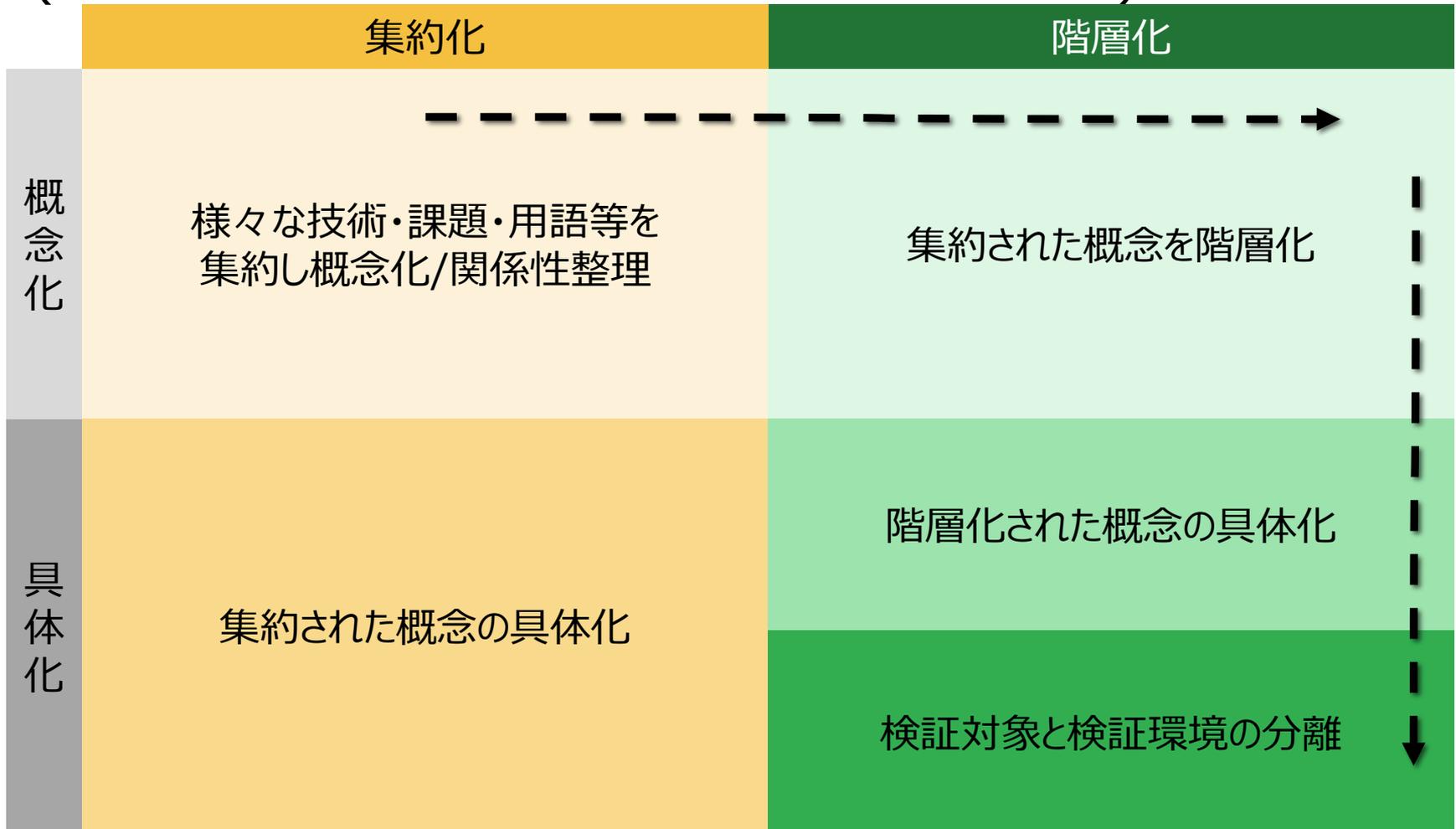
IoTのような複雑なシステムのドメイン分析

- IoT/自動運転システム等の複雑なシステムの構成要素の分類/整理
- これまでの議論/発表を通して出てきた用語から整理を始める



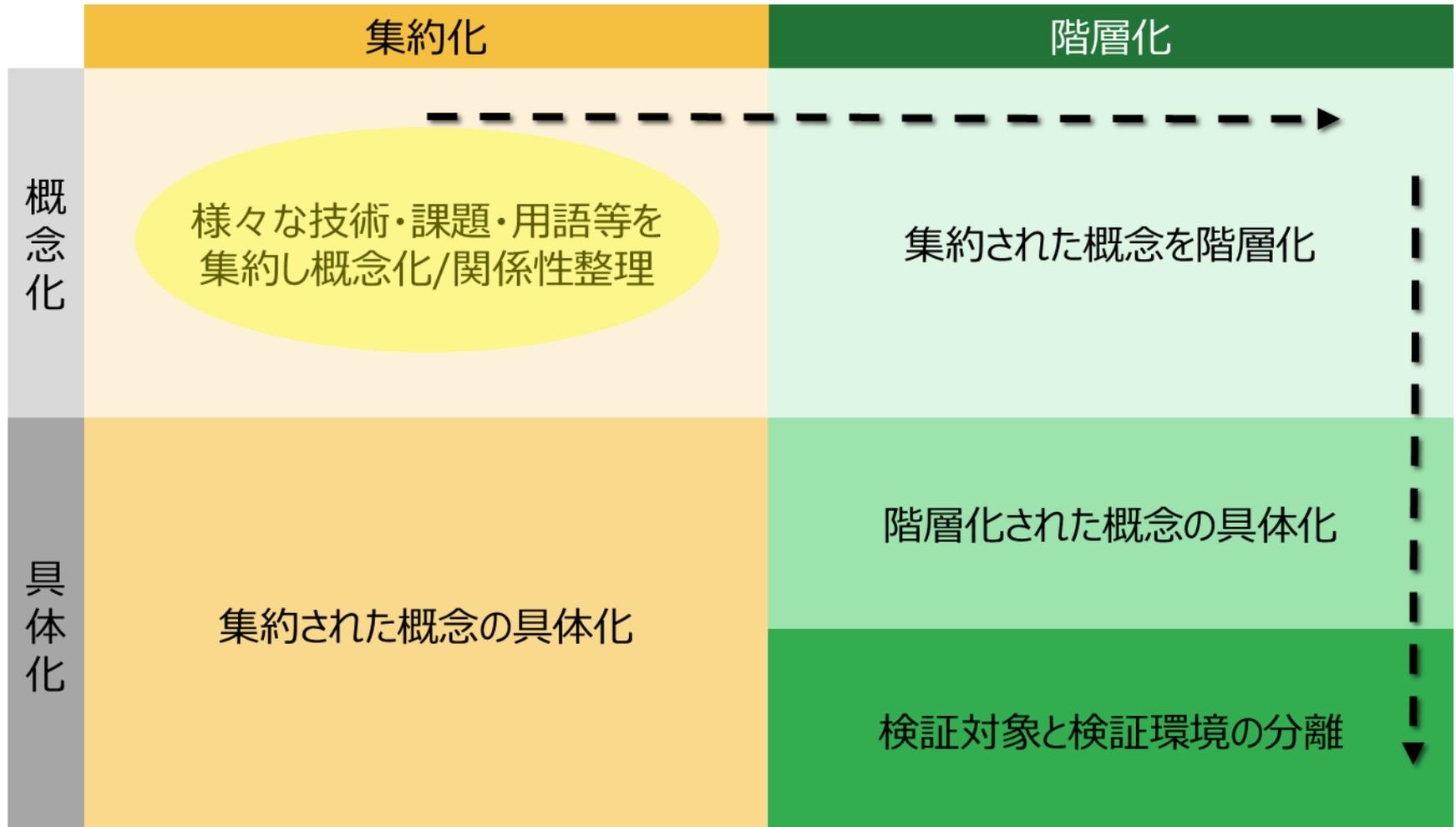
IoTのような複雑なシステムのドメイン分析

- IoT/自動運転システム等の複雑なシステムのドメイン分析方法
(まずは自分たちの知見の範囲で分析することにした)



IoTのような複雑なシステムのドメイン分析

- 集約⇒概念化



これまでの議論/発表を通して出てきた用語

運用アプリ
配車アプリ
自動運転制御
ロボットサービス
スマホ
IoT機器
工場内スペース
センサ
通信
ロボット間通信
ネットワーク系エンジニア
自動運転制御系エンジニア

配車サービス
運用サービス
車両力学モデル
ロボットシステム
クラウド
車載PC
クラウド連携
ロボット
マイコン
ロボット内マイコン群 CAN
バックエンドサーバ系エンジニア

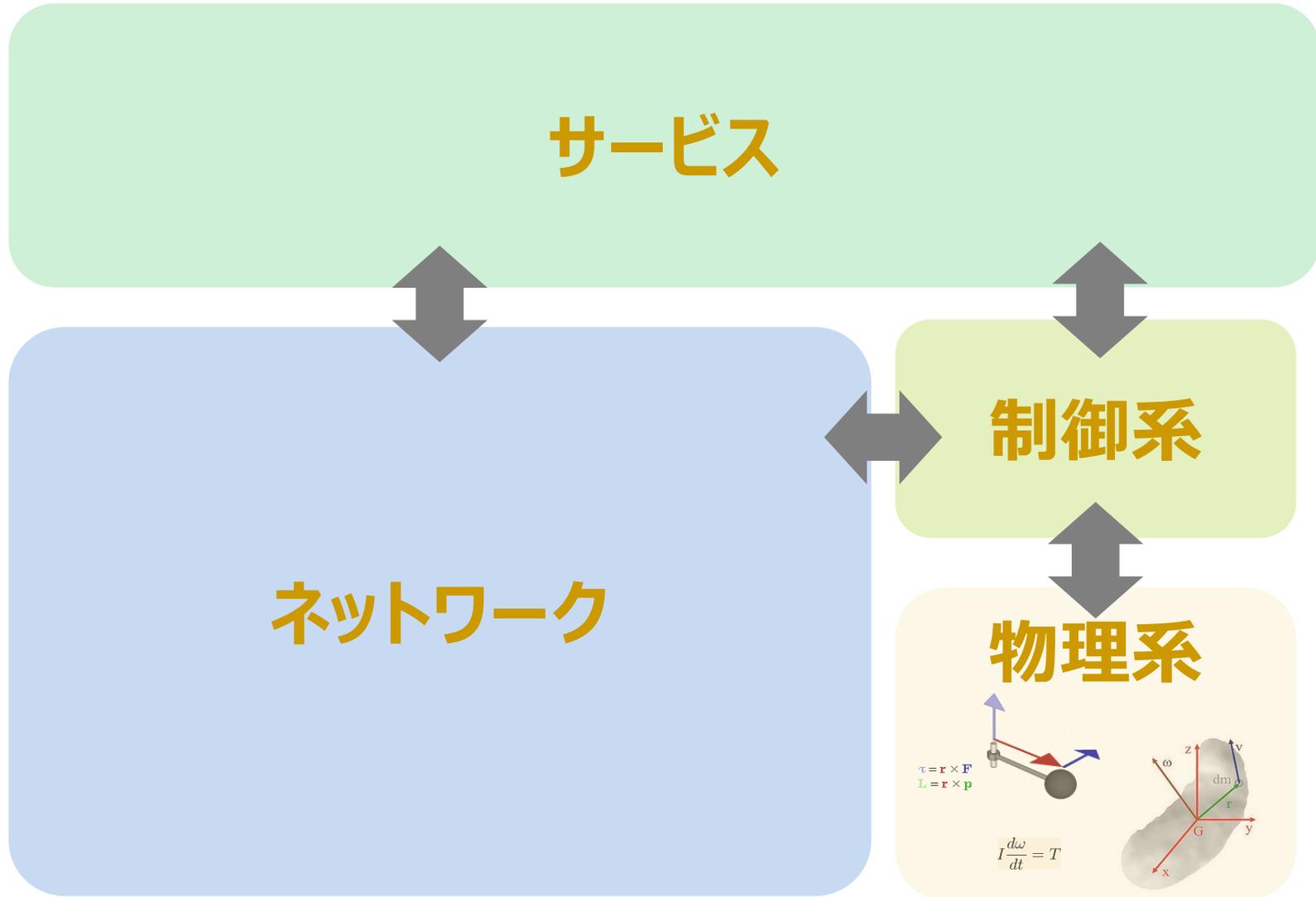
走行環境モデル
ECU制御
歩行者モデル
交通サービス
管制サーバ
車
サーバー
走行環境
ロボット群
Webエンジニア
ECU制御系エンジニア

用語を分類/整理し, 4つの概念に集約

分類	ネットワーク	Web系	バックエンド サーバ	ECU 制御系	自動運転 制御系	メカ系	実証実験
ソフト	—	運用アプリ 配車アプリ	配車サービス 運用サービス ロボットサービス ロボットシステム 交通サービス	ECU制御	自動運転制御	車両力学 モデル	走行環境モデル 歩行者モデル
ハード	クラウド クラウド連携 通信 ロボット間通信 CAN	スマホ	管制サーバ サーバー	IoT機器 センサ マイコン ロボット内マイコン 群	車載PC	車 ロボット ロボット群	工場内スペース 走行環境
エンジニア	ネットワーク系 エンジニア	Web エンジニア	バックエンドサーバ系 エンジニア	ECU制御系 エンジニア	自動運転制御系 エンジニア	—	—

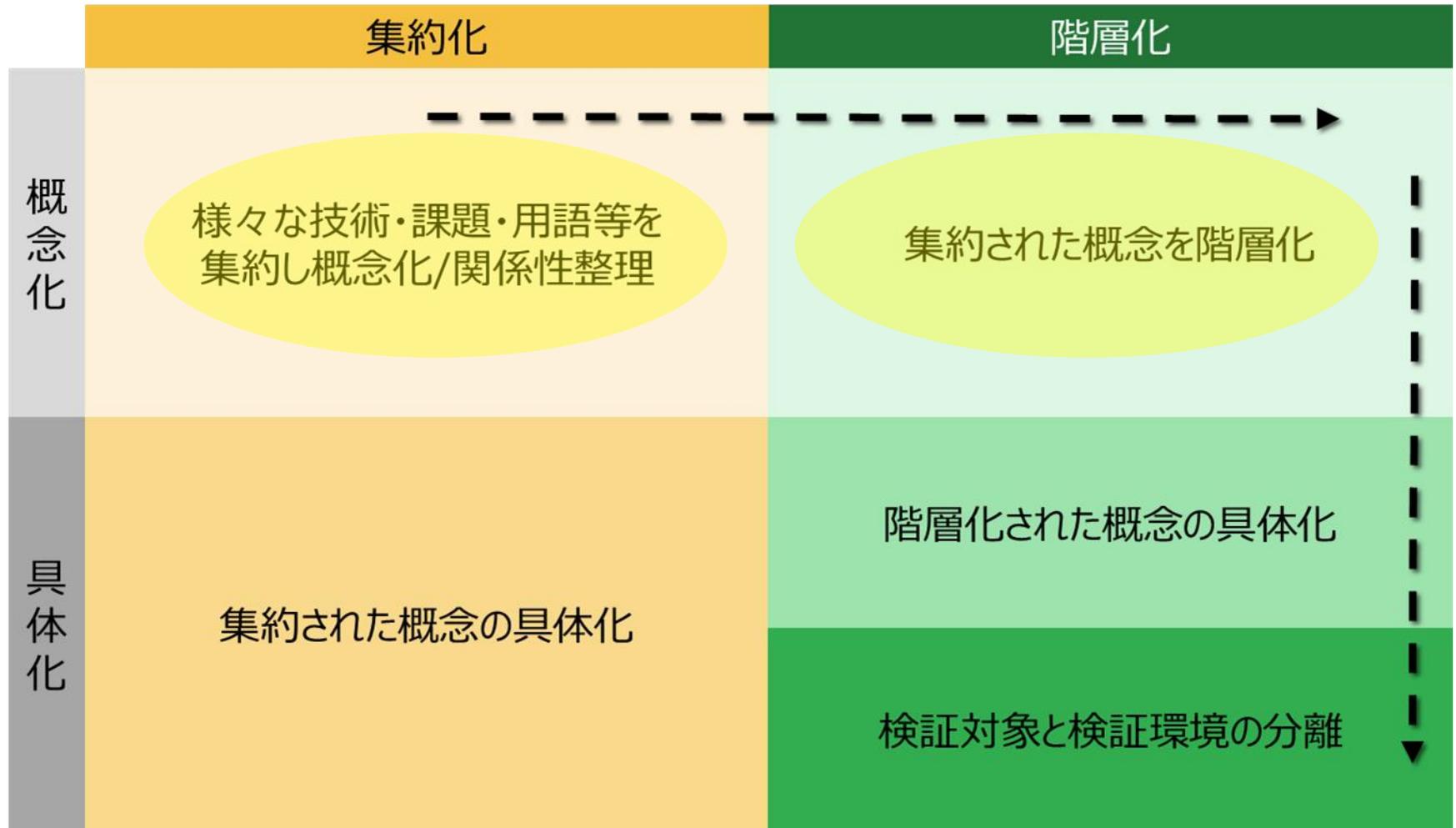


概念間の関係性

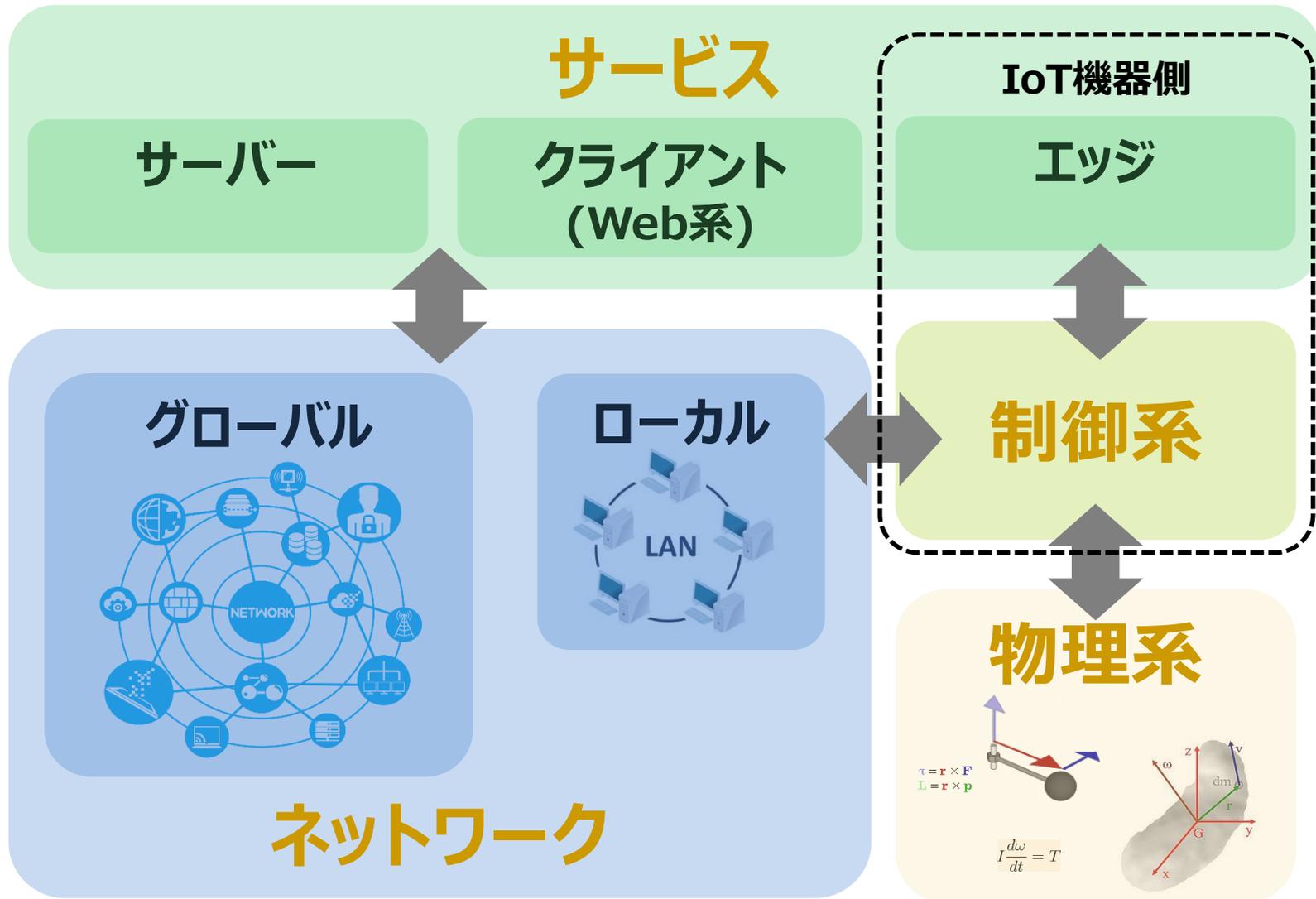


IoTのような複雑なシステムのドメイン分析

- 概念の階層化

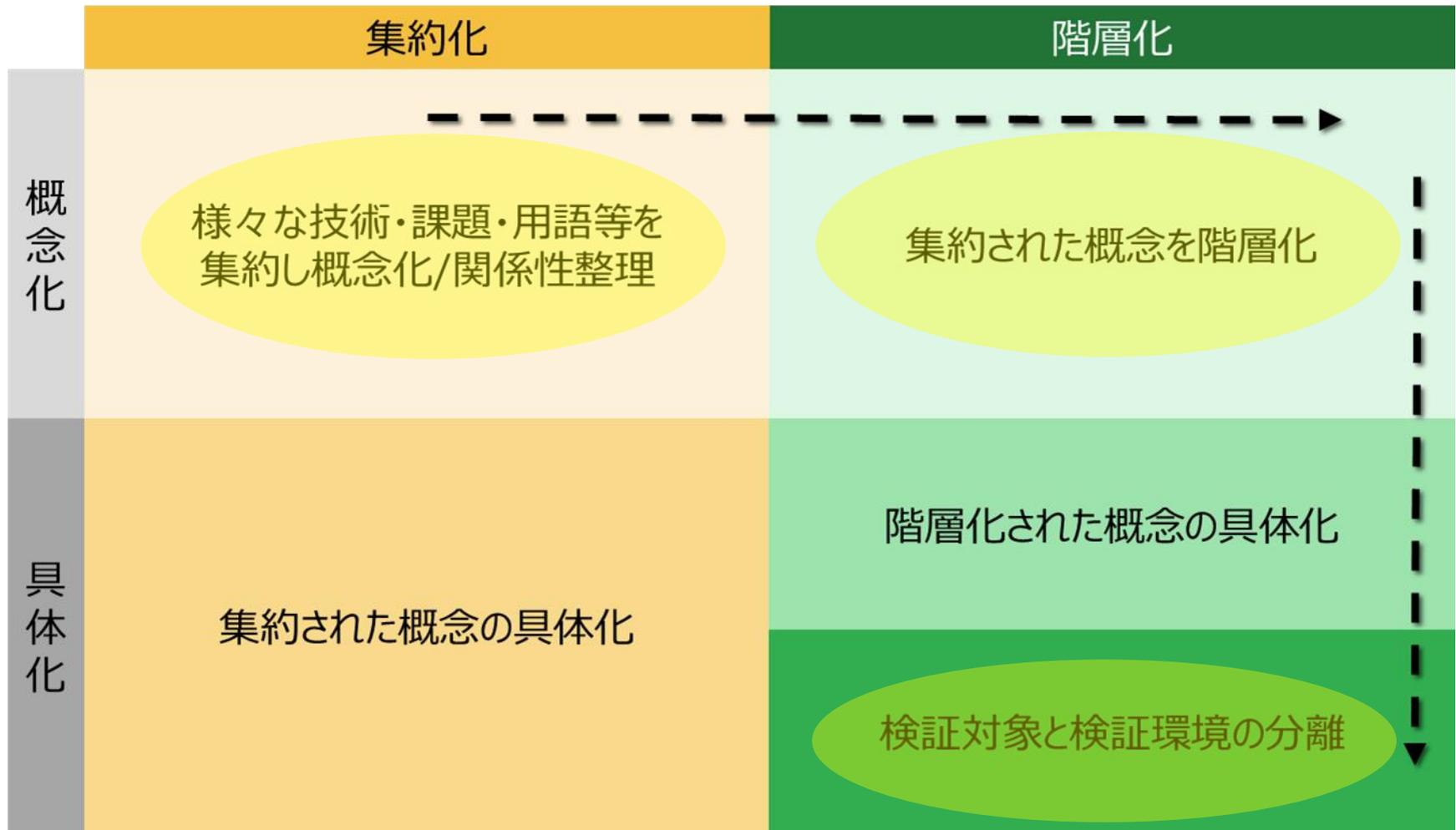


概念の階層化



IoTのような複雑なシステムのドメイン分析

- 検証対象と検証環境の分離



検証対象と検証環境の分離

階層化された概念		検証対象	検証環境	
			ハード	ソフト
サービス	(共通)	提供サービス	(下記全て)	(下記全て)
	サーバー	サーバーソフト	サーバー機器	汎用OS/ミドル/PF
	クライアント	Webアプリ	スマホ	
	エッジ	サービスソフト	IoT機器	RTOS/組込みミドル/PF
制御系		制御ソフト		
物理系	(共通)	—	物理ダイナミクス	—
	周辺環境	—	移動体	—
			実証実験場/ 交通インフラ	—
ネットワーク		—	ネットワーク機器	—

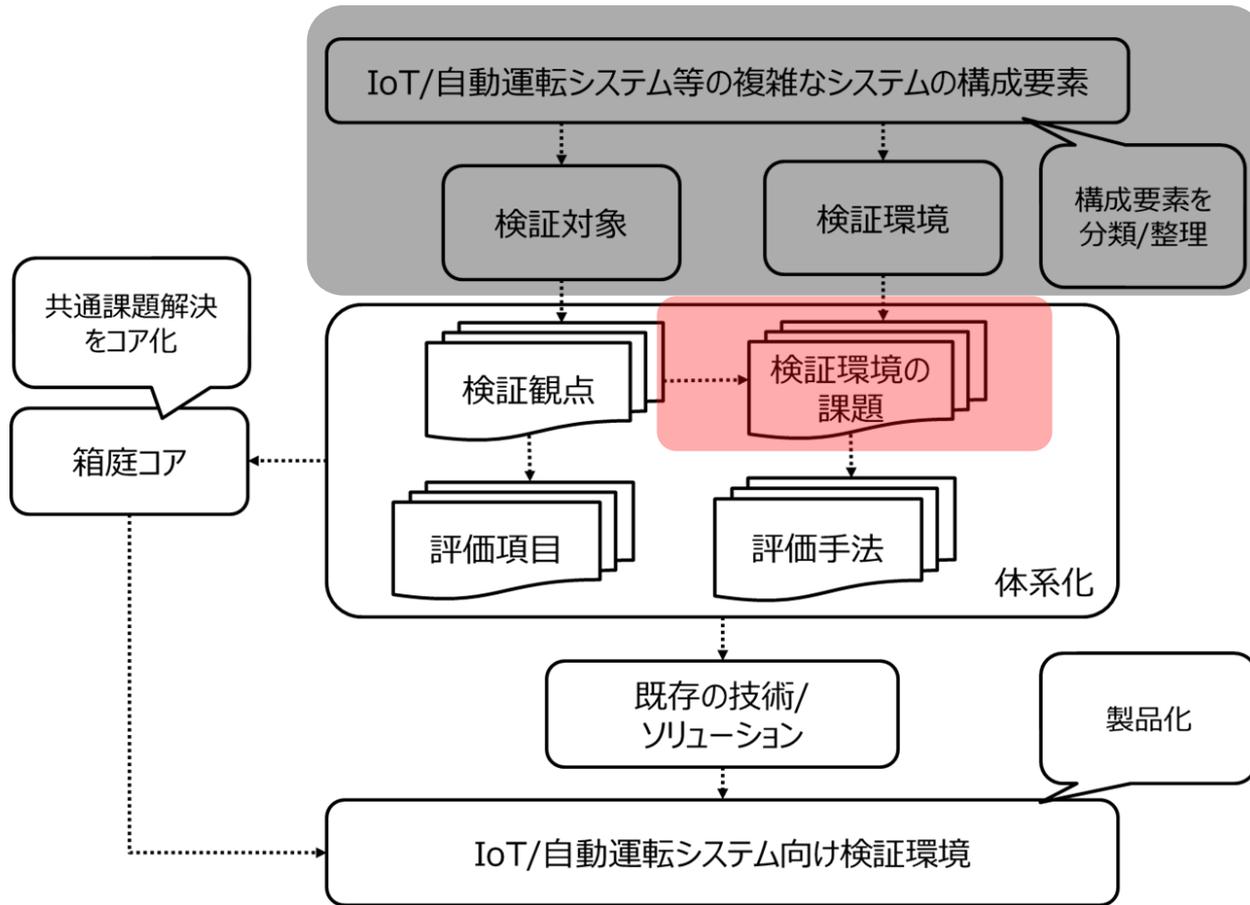
トピック

この分析過程で、2点、気づきがありました(暗黙知⇒表出化)

- 箱庭アセットの正体は、『検証環境』だった
- 検証対象は、『ハード/基盤ソフト上で動作するアプリケーション』だった
(補足：仮想化レベルがL3の場合)

		検証対象 はアプリ	アセット	
階層化された概念		検証対象	検証環境	
			ハード	ソフト
サービス	(共通)	提供サービス	(下記全て)	(下記全て)
	サーバー	サーバーソフト	サーバー機器	汎用OS/ミドル/PF
	クライアント	Webアプリ	スマホ	
	エッジ	サービスソフト	IoT機器	RTOS/組込みミドル/PF
制御系		制御ソフト		
物理系	(共通)	—	物理ダイナミクス	—
	周辺環境	—	移動体	—
		—	実証実験場/ 交通インフラ	—
ネットワーク		—	ネットワーク機器	—

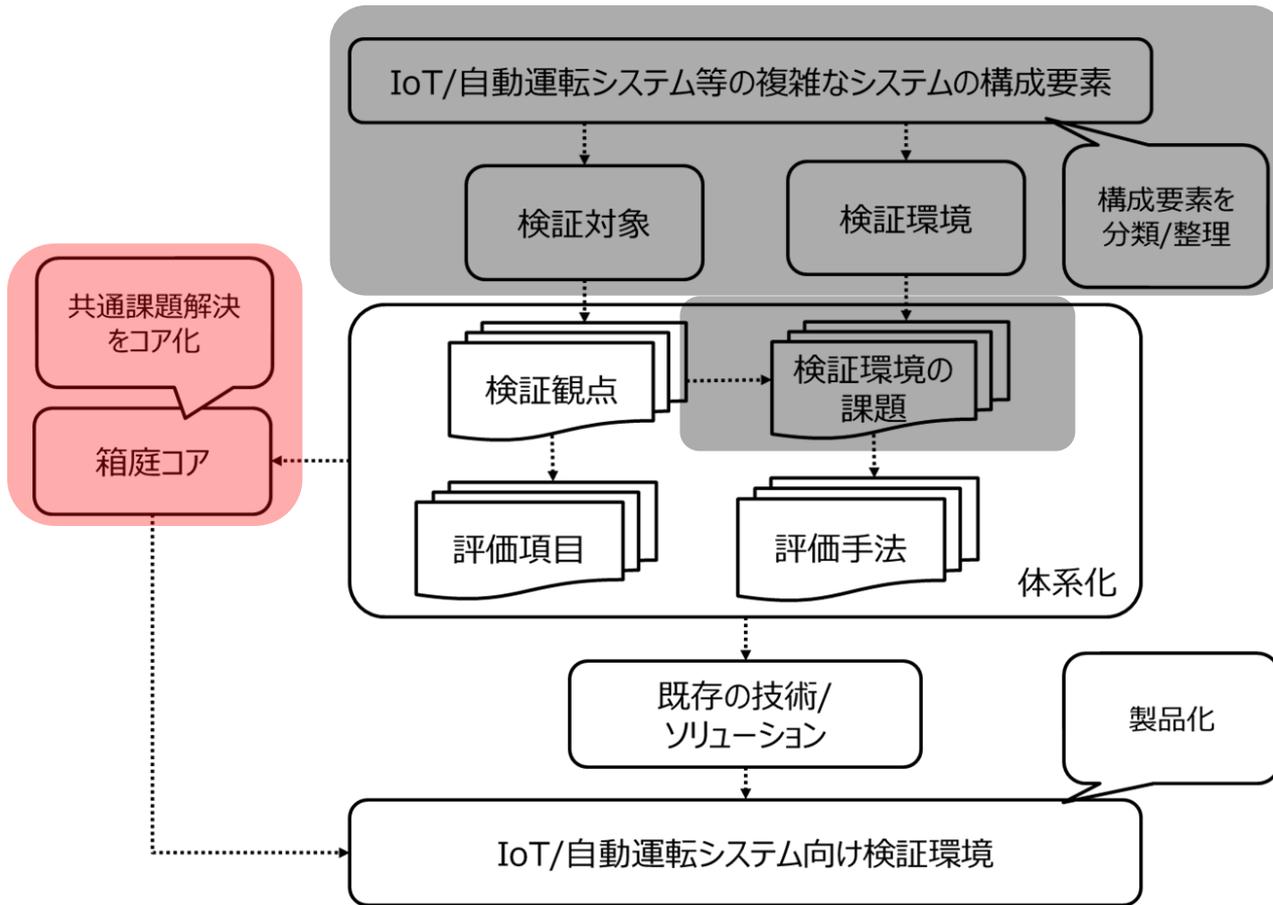
検証環境の課題整理



検証環境の課題

階層化された概念		検証環境		検証環境の課題
		ハード	ソフト	
サービス	(共通)	(下記全て)	(下記全て)	<ul style="list-style-type: none"> • 様々な機器をどう組み合わせるとよいか？ • サービスのイメージ共有(顧客/開発者) • 機器間の不整合が頻発 • トラブル対応の時間・手間・コスト
	サーバー	サーバー機器	汎用OS/ ミドル/PF	
	クライアント	スマホ		
	エッジ	IoT機器	RTOS/ 組込みミドル/ PF	
制御系				<ul style="list-style-type: none"> • 頻繁な仕様変更による再テストの手間 • 最適なRTOS/ミドル/PFの選定 • 機器故障系のテスト
物理系	(共通)	物理ダイナミクス	—	<ul style="list-style-type: none"> • 実証実験場/インフラの確保ができない • 機器故障系は試せない • 不確実要素(人・動物等)の検証は難しい
	周辺環境	移動体	—	
		実証実験場/ 交通インフラ	—	
ネットワーク		ネットワーク機器	—	<ul style="list-style-type: none"> • ネットワーク遅延/切断の影響見極め • 最適なネットワーク機器構成見極め • ネットワーク内パケット解析

箱庭コア機能特性と課題対応状況



箱庭コア機能特性と課題対応状況

概念	検証環境の課題	箱庭コアの機能特性			
		コンポーネント化	可視化	イベント駆動化	自動化
サービス	・様々な機器をどう組み合わせるとよいか？	○	—	—	—
	・サービスのイメージ共有(顧客/開発者)	—	○	—	—
	・機器間の不整合が頻発	—	○	—	—
	・トラブル対応の時間・手間・コスト	—	○	—	—
制御系	・頻繁な仕様変更による再テストの手間	—	—	—	○
	・最適なRTOS/ミドル/PFの選定	○	—	—	—
	・機器故障系のテスト	—	—	○	—
物理系	・実証実験場/インフラの確保ができない	○	—	—	—
	・機器故障系は試せない	—	—	○	—
	・不確実要素(人・動物等)の検証は難しい	○	—	○	—
ネットワーク	・ネットワーク遅延/切断の影響見極め	—	—	○	—
	・最適なネットワーク機器構成見極め	○	—	—	—
	・ネットワーク内パケット解析	—	○	—	—

サマリ

今回試行したこと

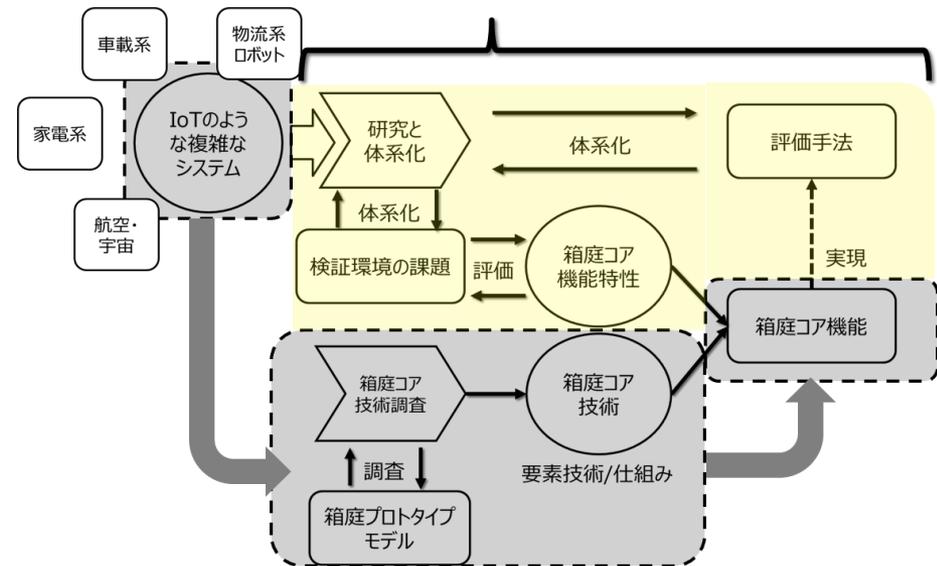
- 箱庭コア技術検討にあたり，IoTのような複雑なシステムの「検証課題」や「評価手法」も整理する必要があることに気づき，技術調査の進め方を見直した
- まずは，これまでの知見をベースに，かけ足で検証課題を導き，箱庭コア機能特性の充足性をチェックできた

気づいたこと

- 漠然と捉えていたものが，概念とその関係性として整理された
- 検討が不足している点も見えてきた（ネットワーク仮想化等）

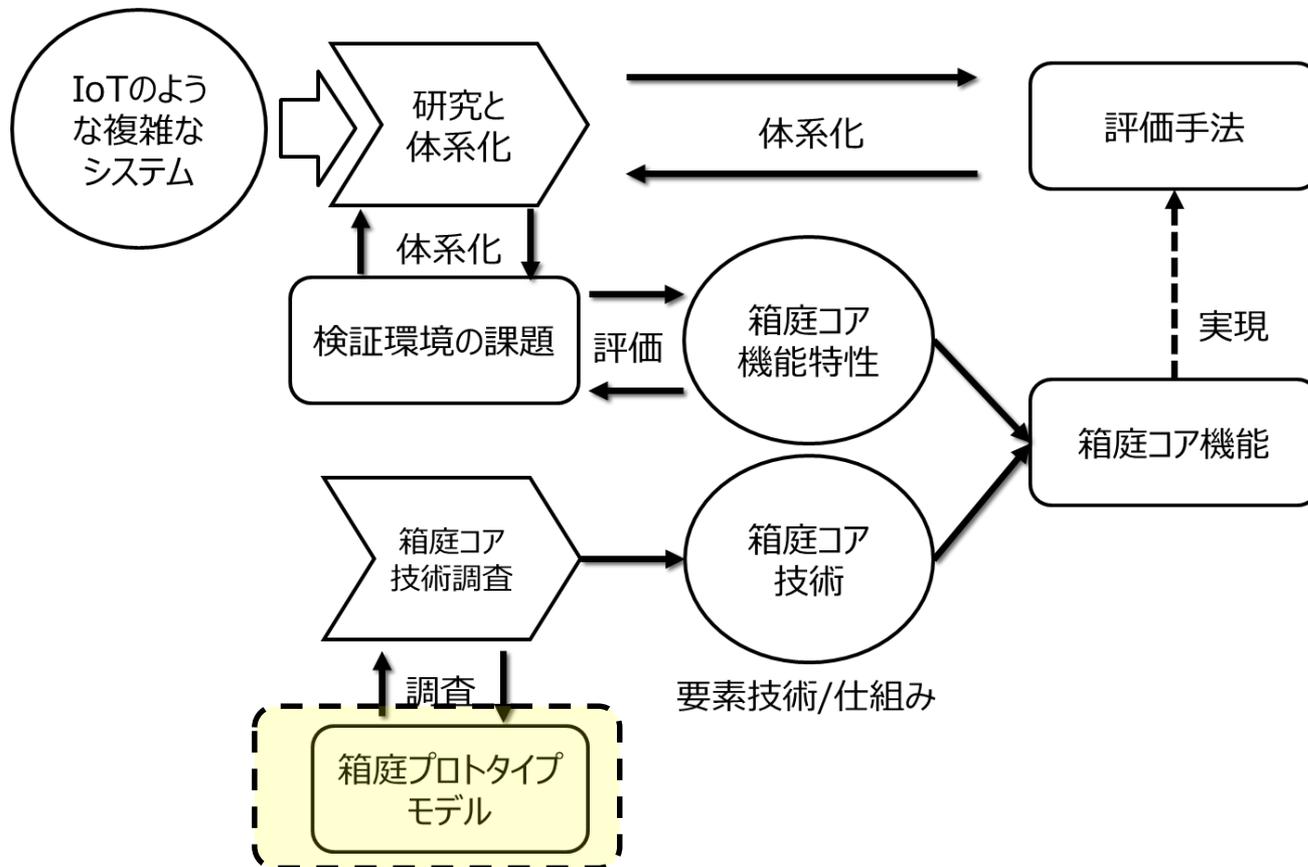
今後の課題

- 駆け足での分析だったので様々な分野の方からのFBをもらいたい
- 体系化していくためにも要件管理&腰を据えが分析が必要

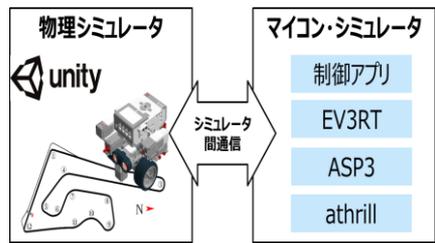
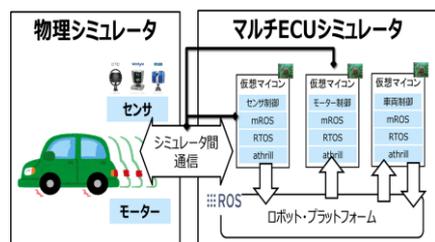
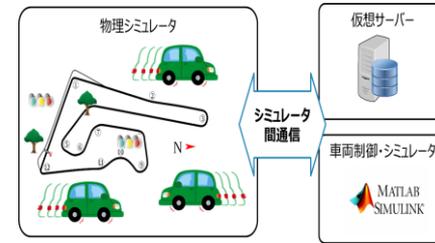


箱庭プロトタイプモデルの開発状況

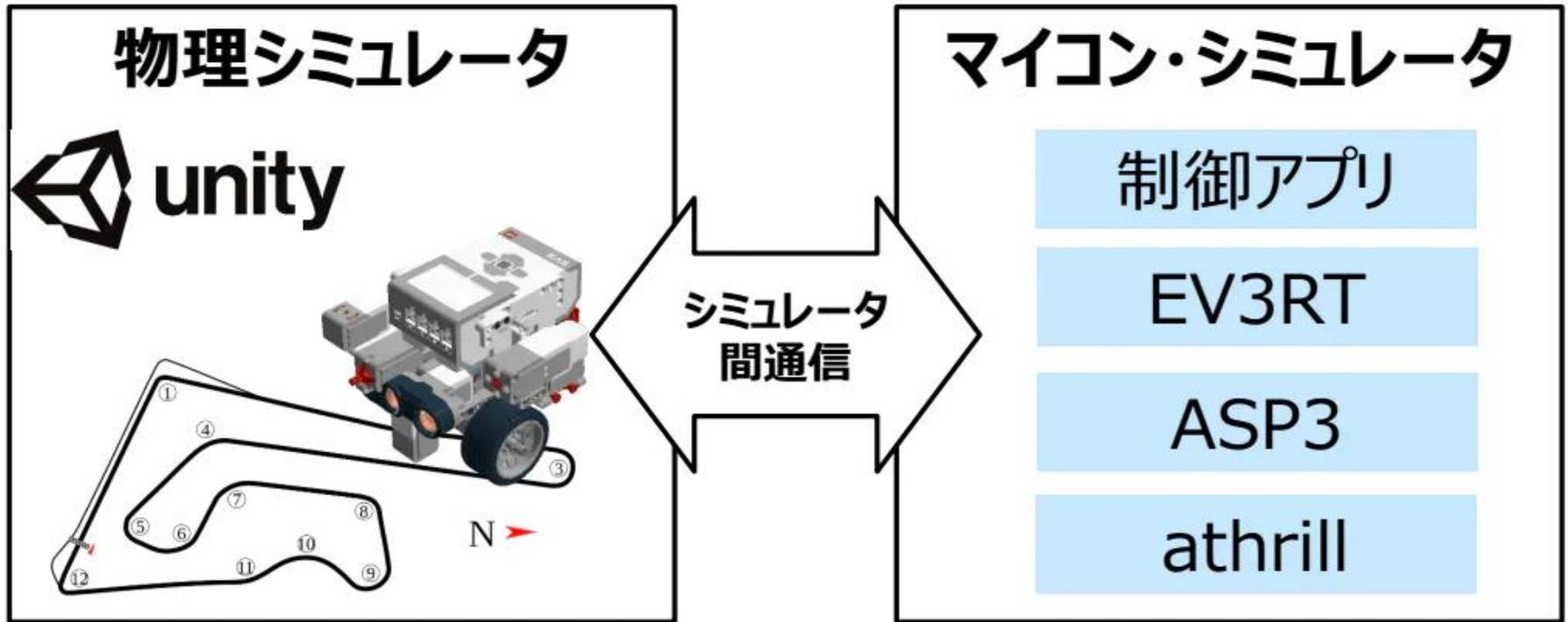
- 箱庭プロトタイプモデルの開発状況
- ETロボコンシミュレータ開発の進捗状況



箱庭プロトタイプモデルの開発状況

プロトタイプモデル	技術調査	実装	動作確認	配布
	○	○	○	×
<p> 1 ターン目 : ○ : ライトレース 2 ターン目 : × : 超音波センサ/ジャイロセンサ 3 ターン目 : × : その他(要望に応じて随時対応) </p>				
	○	○	△	×
<p> 実装までは完了 ECU間の通信ができない現象が発生しており、デバッグ中... </p>				
	×	×	×	×
<p> 未着手 上記シミュレータの目途ができてから着手予定 </p>				

ETロボコン・シミュレータ構成



ETロボコンの検証対象と検証環境

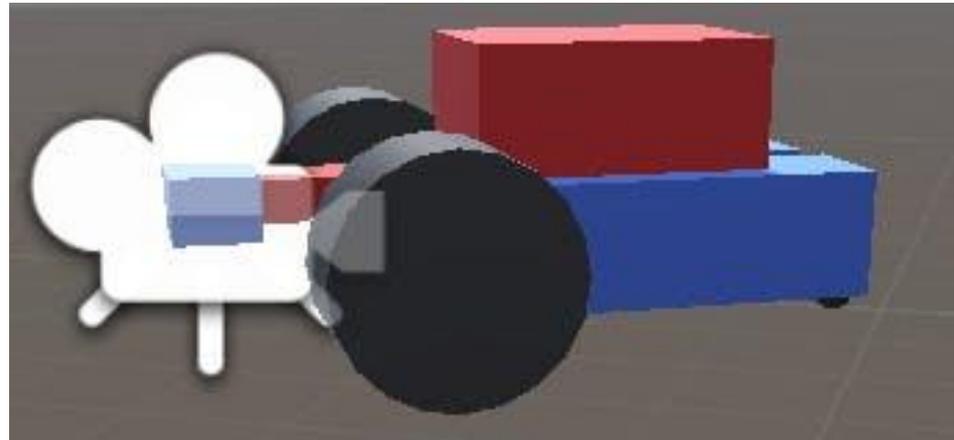
階層化された概念		検証対象	検証環境	
			ハード	ソフト
サービス	(共通)	(なし)	(なし)	(なし)
	サーバー	(なし)	(なし)	(なし)
	クライアント	(なし)	(なし)	
	エッジ	(なし)	EV3	
制御系	制御ソフト	<ul style="list-style-type: none"> - サーボモータ - カラーセンサ - 超音波センサ 		
物理系	(共通)	—	物理ダイナミクス	—
	周辺環境	—	走行コース	—
			照明	—
ネットワーク		—	Bluetooth機器	—

ETロボコン検証環境の課題

概念	検証環境の課題	ETロボコン検証環境の課題	
		有無	具体的な課題
サービス	・様々な機器をどう組み合わせるとよいか？	×	—
	・サービスのイメージ共有(顧客/開発者)	○	開発者間の走行制御方式の共有
	・機器間の不整合が頻発	×	—
	・トラブル対応の時間・手間・コスト	○	走行トラブル時のデバッグ
制御系	・頻繁な仕様変更による再テストの手間	○	走行制御仕様の頻繁な変更
	・最適なRTOS/ミドル/PFの選定	×	—
	・機器故障系のテスト	○	照明の強さによる影響テスト
物理系	・実証実験場/インフラの確保ができない	○	走行コースの確保
	・機器故障系は試せない	○	照明のセッティング
	・不確実要素(人・動物等)の検証は難しい	×	—
ネットワーク	・ネットワーク遅延/切断の影響見極め	○	Bluetoothの通信遅延等
	・最適なネットワーク機器構成見極め	×	—
	・ネットワーク内パケット解析	○	Bluetooth内パケット解析

検証環境(ハード) : EV3

- センサ
 - カラーセンサ
 - アクチュエータ
 - サーボモータ(L, R)
 - 寸法
 - 実物の10倍(※).
- (※) 例 : タイヤ幅は実物約3cmですが, 30cmにしています.



検証環境(ハード) : 物理ダイナミクス

- 物理ダイナミクス
 - 物理シミュレータ
 - Unity
 - シミュレーション時間
 - 1msec
 - EV3のモーター/ホイールの特徴
 - 走行方向変更は, 自動車のようにホイールのステアリング角で制御しない
 - ホイールの回転数の差でステアリング角制御する

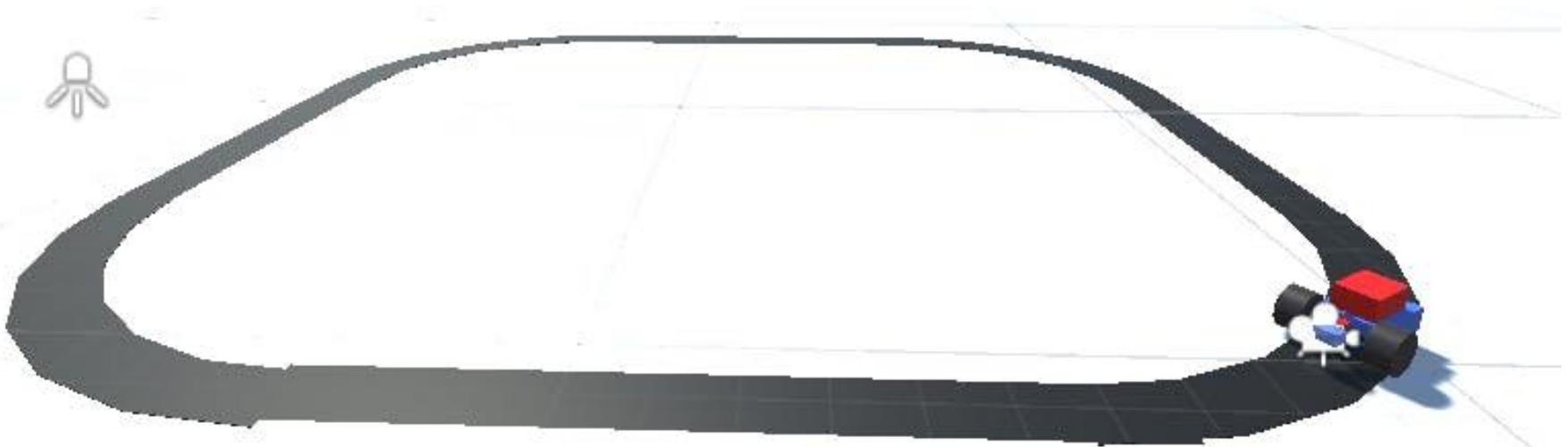


- EV3のモーター/ホイールの実現方法
 - Unityの標準組込みのダイナミクス(HingeJoint)を利用
 - ドアなどの蝶番用のダイナミクスであり, 車軸を中心に回転させるもの
 - <https://docs.unity3d.com/ja/current/ScriptReference/HingeJoint.html>



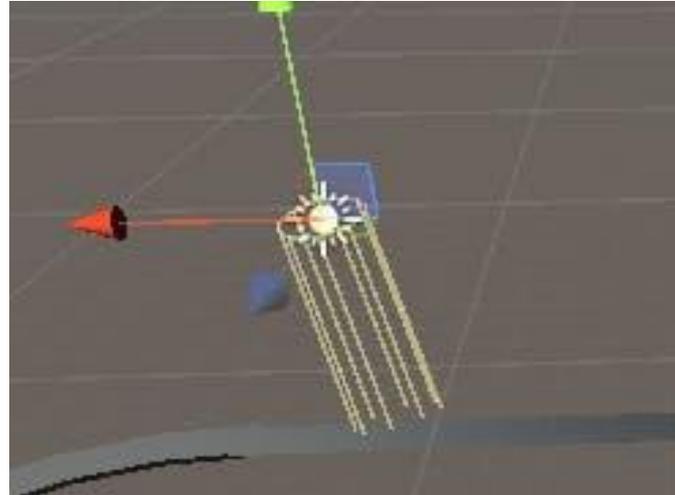
検証環境(ハード) : 走行コース

- まずは単純なコースから

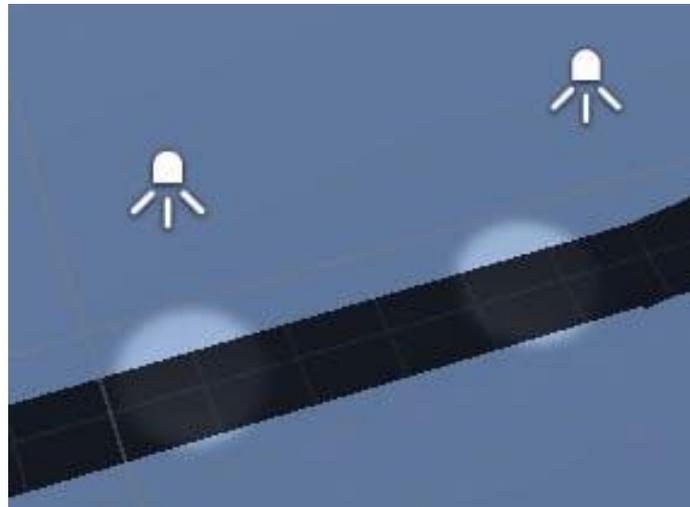


検証環境(ハード) : 照明

- 全体照明 : 1 個

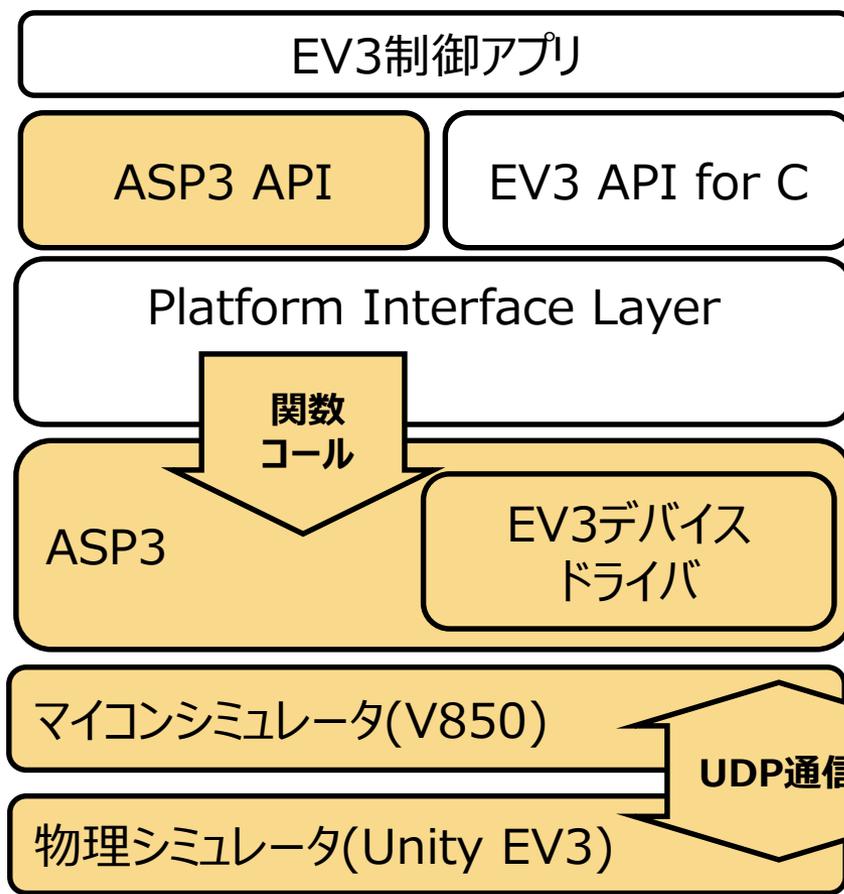
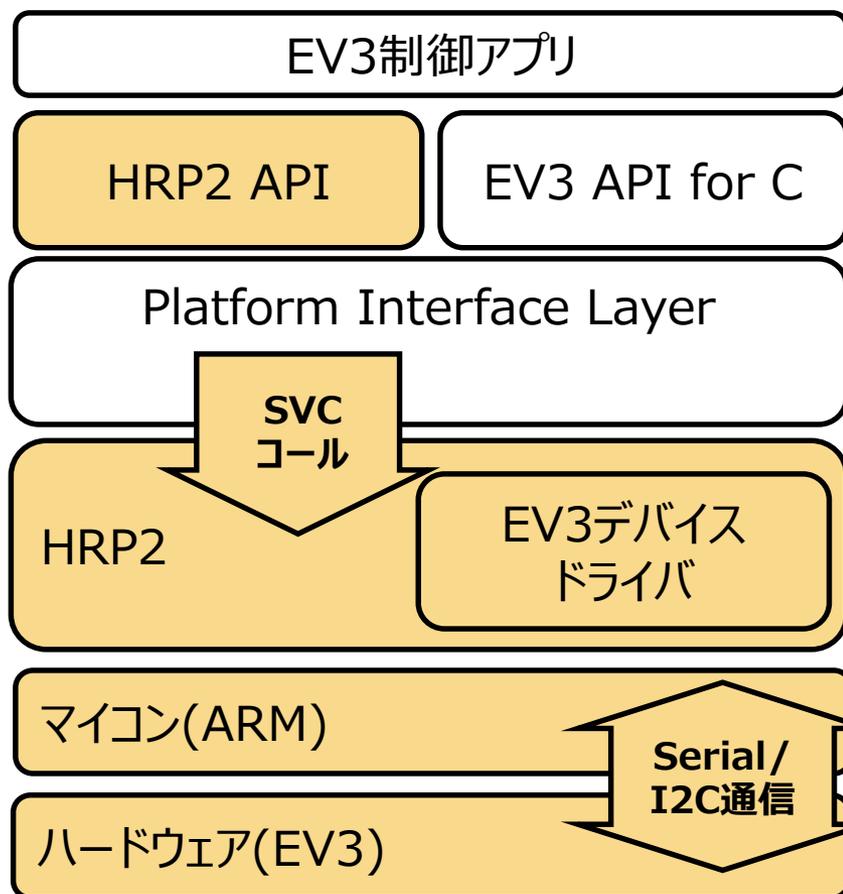


- スポットライト : 2 個



検証環境(ソフト) : EV3RT/ASP3/デバドラ

- EV3制御アプリレベルでは同じソースコードがそのまま利用できる

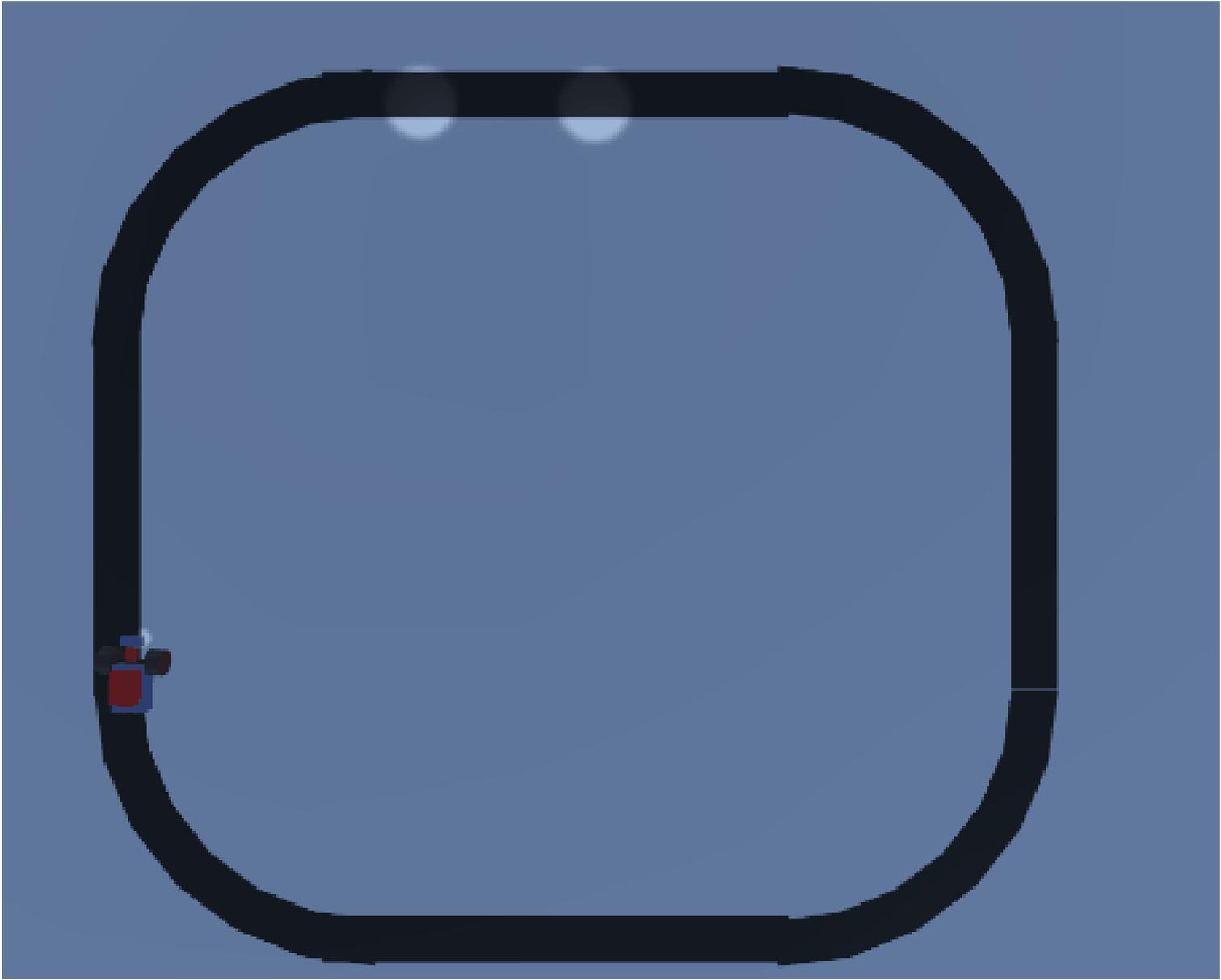


検証対象：制御ソフト

- ライントレース制御
 - main_task : 100msec周期制御

```
while(1) {  
  
    /**  
     * PID controller  
     */  
#define white 100  
#define black 10  
    static float lasterror = 0, integral = 0;  
    static float midpoint = (white - black) / 2 + black;  
    {  
        float error = midpoint - ev3_color_sensor_get_reflect(EV3_PORT_1);  
        integral = error + integral * 0.5;  
        float steer = 0.07 * error + 0.3 * integral + 1 * (error - lasterror);  
        ev3_motor_steer(left_motor, right_motor, 10, steer);  
        lasterror = error;  
    }  
    tslp_tsk(100000); /* 100msec */  
  
}
```

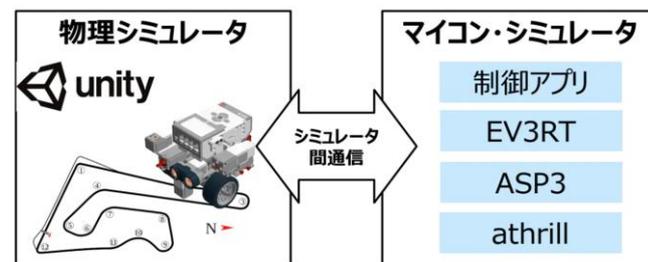
デモ



サマリ

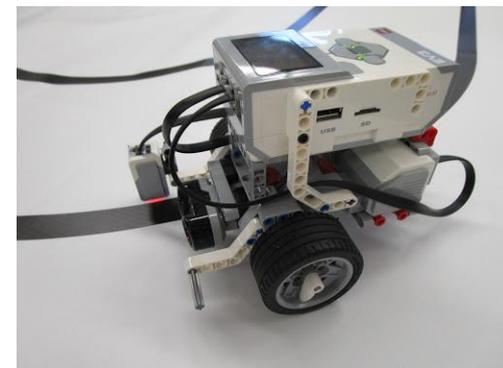
● 今回試行したこと

- ETロボコンシミュレータを駆け足で作りました
- ライントレースは出来るようになりました
- qiita記事でインストール手順を公開しました



● 気づいたこと

- シミュレータでは動作したものの、以下の視点で、現実的に使えるものになったのかがわからない
 - 機能範囲, センサ精度, 使い勝手等



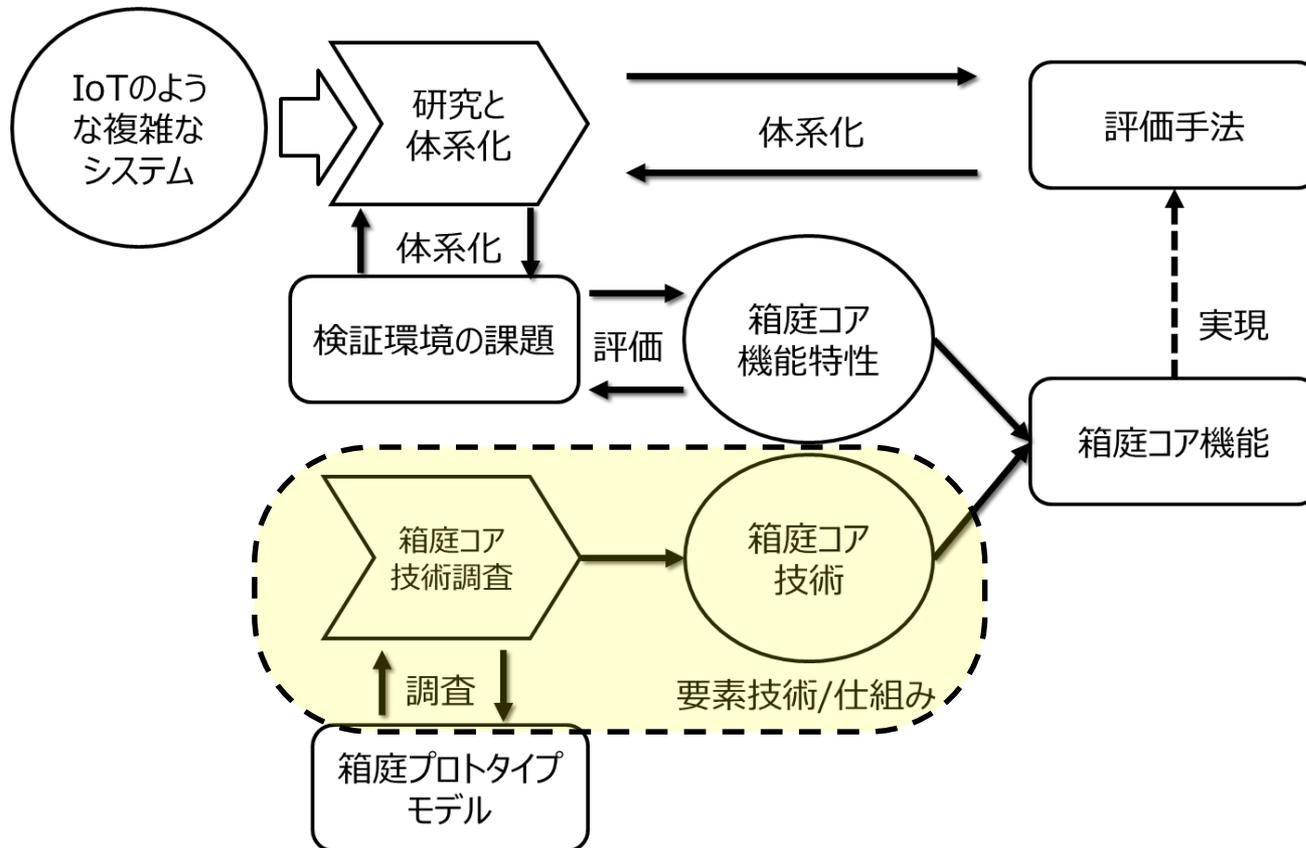
● 今後の課題

- 実物のEV3でのシミュレーション精度/評価を行う
- ETロボコン出場者向けに利用者層を増やしていく
⇒FBもらって箱庭コア技術の知見を広げる



箱庭コア技術の検討状況

- シミュレーション時間同期
 - 基本概念の整理
 - シミュレーション時間同期の検討状況



シミュレーション時間同期の基本概念的整理

• シミュレータ

- マイコン・シミュレータ
- 物理シミュレータ
- ネットワーク・シミュレータ

• 時間

- 仮想時間/シミュレーション時間
- 離散時間ステップ
- 同期離散時間ステップ
- 実時間
- シミュレーション時間同期
 - 完全同期
 - 非完全同期(遅延幅予測可能)
 - 非完全同期(遅延幅観測可能)

マイコン・シミュレータ

- マイコンの挙動を再現するシミュレータ.
- 再現レベルは様々なバリエーションがあり, 少なくとも以下がある.
 - **CPUの再現レベル:**
 - CPU-L1.
 - クロックサイクルレベルでの再現(RTOS/デバイスドライバ開発者向け)
 - CPU-L2.
 - 命令セットレベルでの再現(RTOS/デバイスドライバ開発者向け)
 - CPU-L3.
 - RTOS/デバイスドライバのI/F仕様レベルでの再現(アプリケーション開発者向け)
 - **周辺デバイスの再現レベル:**
 - DEV-L1.
 - クロックサイクルレベルでの再現(RTOS/デバイスドライバ開発者向け)
 - DEV-L2.
 - ハードウェア・レジスタ仕様レベルでの再現(RTOS/デバイスドライバ開発者向け)
 - DEV-L3.
 - RTOS/デバイスドライバのI/F仕様レベルでの再現(アプリケーション開発者向け)

マイコン・シミュレータの再現レベル

現時点では、箱庭のマイコンシミュレータの再現レベルは以下を想定。

- CPU-L3
- DEV-L3

CPUの再現レベル



周辺デバイスの再現レベル



物理シミュレータ

- 物理演算エンジン
- 「運動」「落下」「反射」「相互作用」「慣性」「摩擦」といった剛体力学系のシミュレーション・エンジン
- TODO事項
 - シミュレーション精度のレベルの定義については未調査

ネットワーク・シミュレータ

- まだ調べきれれておりません・・・

時間

• 仮想時間/シミュレーション時間

- シミュレーション内の時間(離散時間).

• 離散時間ステップ

- シミュレータ内の離散時間の間隔.

• 同期離散時間ステップ

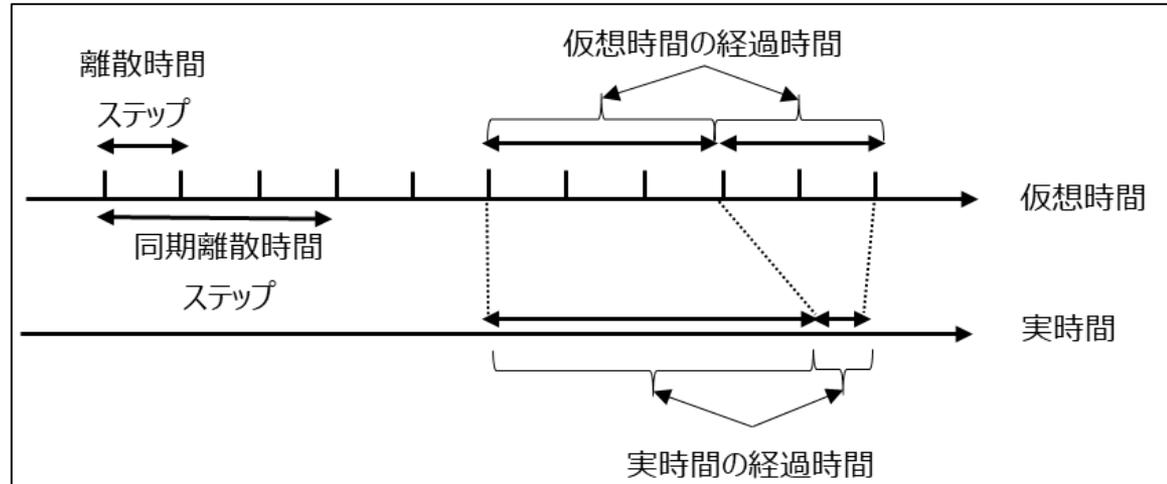
- シミュレータ間で仮想時間を同期する
- 離散時間の間隔(離散時間ステップの整数倍).

• 実時間

- シミュレーション実行時の現実時間.
- 補足 :
- シミュレータを非リアルタイムなシステムで実現する場合,
- シミュレータの実行周期タイミングはばらつく可能性がある.
- そのため, 仮想時間の経過時間と実時間の経過時間は一致しない.

• シミュレーション時間同期

- IoTのような大規模なシミュレーションの場合, 複数のシミュレータを利用する必要がある.
- 一方で, 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れる可能性がある. シミュレーション時間同期は, 各シミュレータの実行タイミングを適切なタイミングに調整するための方法であり, 各シミュレータのシミュレーション時間を同期するものである.



シミュレーション時間同期方式

- シミュレーション時間同期の方式として、以下の3種類がある。

- 完全同期**

- 各シミュレータのシミュレーション時間の進みが、同期離散時間ステップ単位で完全に同期するもの。

- 非完全同期(遅延幅予測可能)**

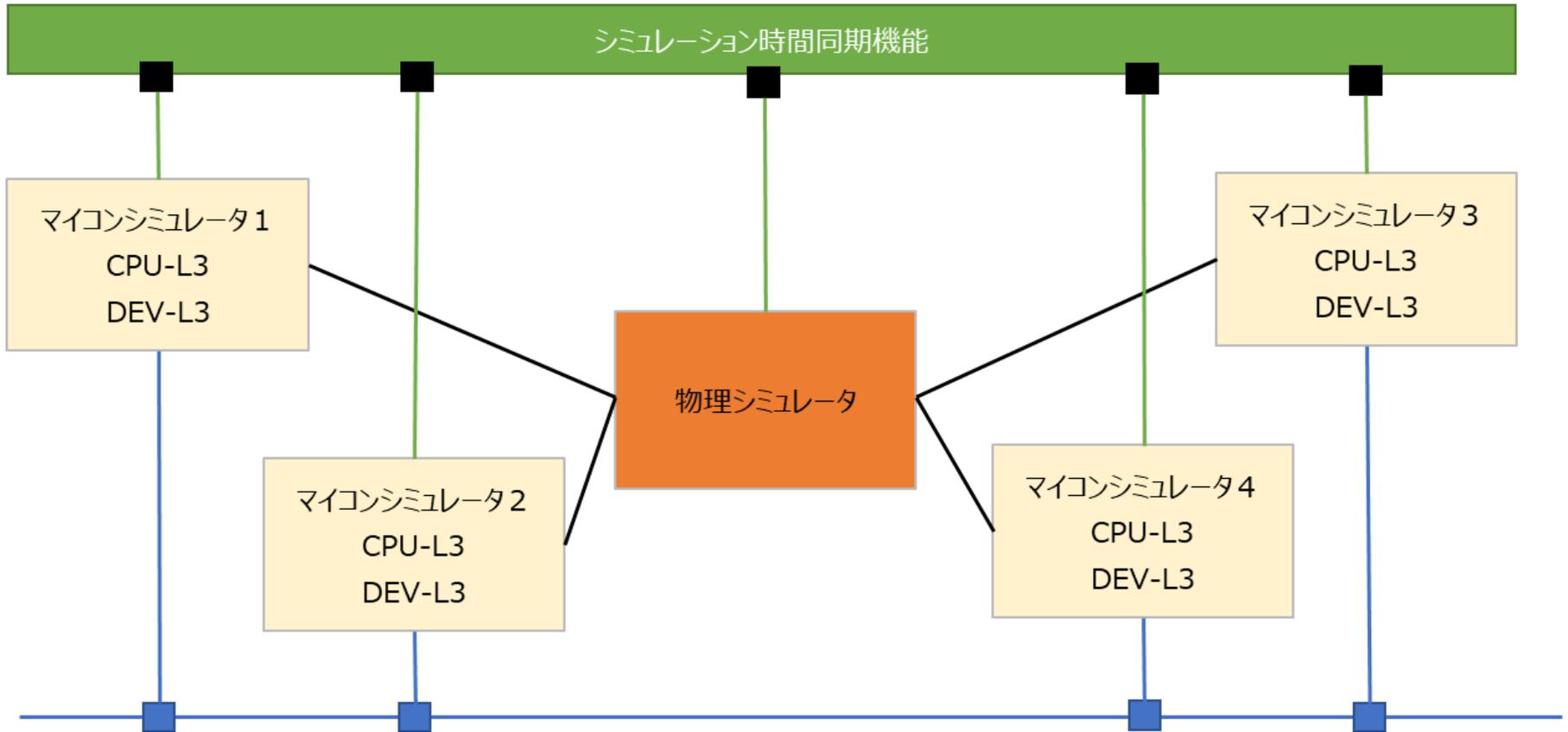
- 各シミュレータのシミュレーション時間の進みは一致しないが、遅延の幅は予測可能なもの。
- (T.B.D) ユースケースがよくわからない。

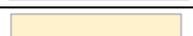
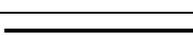
- 非完全同期(遅延幅観測可能)**

- 各シミュレータのシミュレーション時間の進みは一致しないが、遅延幅を低減/調整する仕組みがあるもの。遅延幅の予測はできないが、シミュレーション実行後にどの程度の遅延があったかを観測可能にする仕組みがある。
- 他方式に比べて、各シミュレータの同期コストは低減させやすいため、シミュレーション実行速度は高速化させやすい。そのため、シミュレーション実行環境のスペックを十分に高速化させることで、期待する精度のシミュレーションを可能にし、シミュレーション実行後に適切な精度での実行ができたかどうかの評価も可能となる。

シミュレーション時間同期の実現方式検討

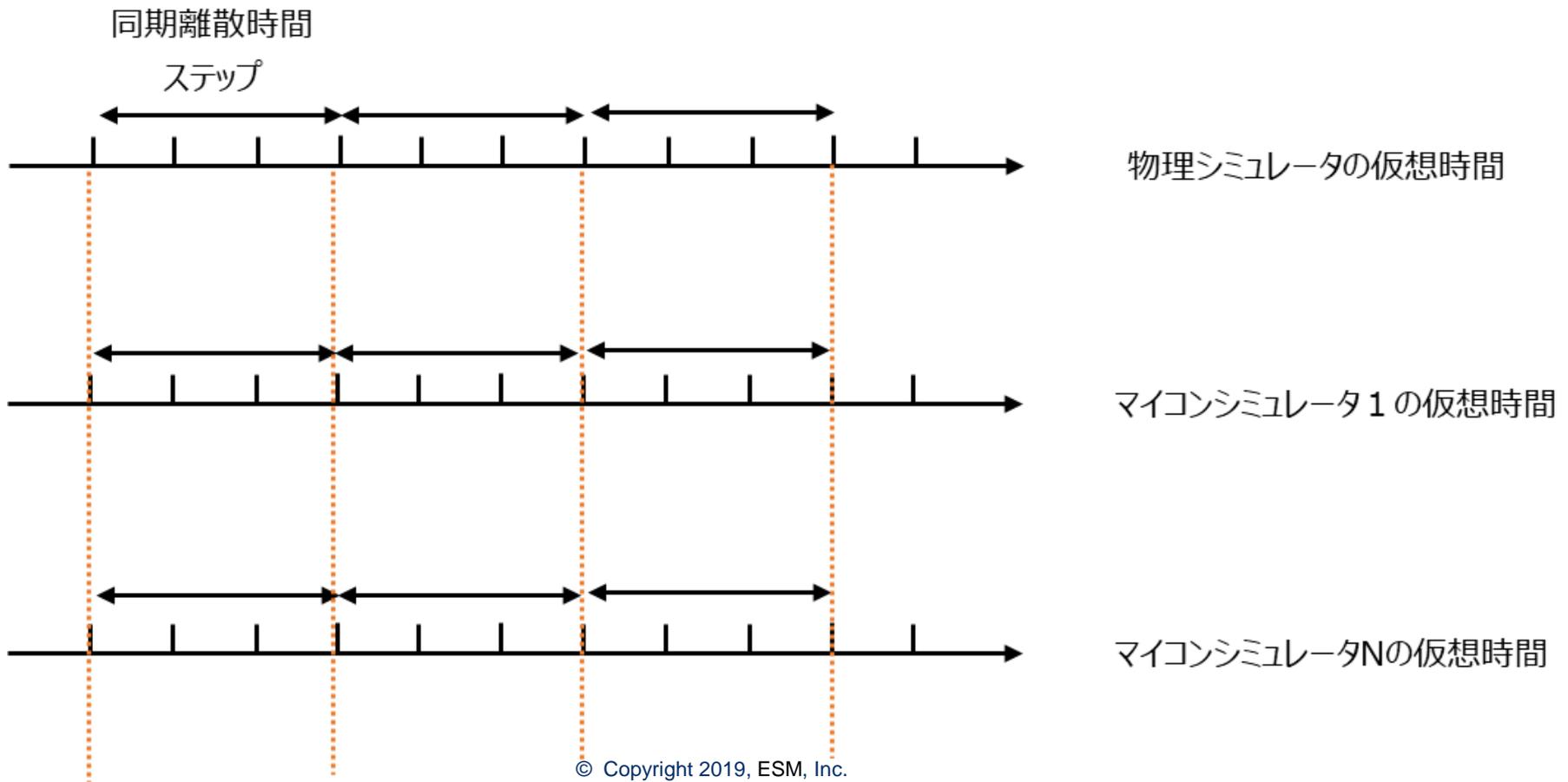
・前提とする構成



構成要素	個数	再現レベル	凡例
シミュレーション時間同期機能	1	-	
物理シミュレータ	1	-	
マイコンシミュレータ	N	CPU-L3, DEV-L3	
シミュレーション時間同期機能との通信インフラ	N	-	
物理シミュレータ・マイコンシミュレータ間の通信インフラ	N	-	
マイコンシミュレータ間の通信インフラ	M	-	

実現方式(完全同期)

- 物理シミュレータのとマイコンシミュレータ間で適切な「同期離散時間ステップ」を決め、パラレルまたはシーケンシャルに、物理シミュレータ/マイコンシミュレータのシミュレーション時間を同期離散時間ステップだけ進める。
 - (同時実行の場合は、全シミュレータの終了を待ち合わせする。)



同期離散時間ステップを決める際に考慮すべきポイント

• 同期離散時間の値

- 各シミュレータの離散時間ステップの最小公倍数の倍数とすること

• 通信順序

- マイコンシミュレータ間の通信データの到着順を保証すること

• オーバーヘッド

- 同期離散時間ステップが小さいとシミュレータ間の同期通信(プロセス間通信)コストが高くなるため、求めるシミュレーション精度とシミュレーション全体の实時間から適切な値を選択すること

非完全同期(遅延幅予測可能)

- まだ検討できていません。
※FMIの仕様が参考になると思われます

非完全同期(遅延幅観測可能)

• 考え方 :

1. 物理シミュレータのシミュレーション時間を基準にして、マイコンシミュレータの仮想時間を同期離散時間ステップの周期で同期させる.
2. 物理シミュレータとマイコンシミュレータの仮想時間の遅延幅は、物理シミュレータ側で観測可能にする.

非完全同期(遅延幅観測可能)

- 本実現方式の前提事項として、以下のものがある。
 - マイコン・シミュレータの機能・性能
 - マイコンシミュレータの実行速度(実時間レベルでの仮想時間の進む速度)が物理シミュレータに比べて早いこと。
 - マイコンシミュレータの実行速度を調整するための前提条件および機能を有すること
 - 前提：マイコンシミュレータ上で動作するソフトウェアのCPU使用率は低いこと。
 - 機能：CPUがHALT状態になった場合、仮想時間の進み幅を調整する機能があること。
 - シミュレーション精度の制約
 - 物理シミュレータとマイコンシミュレータの仮想時間の遅延幅は、同期離散時間ステップ以上になる場合がある。
 - 物理シミュレータは仮想時間の通知のみであるため、高速に動作するが、シミュレータ間の時間同期および予測可能な遅延幅を保証しない。

同期方法(物理シミュレータ)

• 実現方法

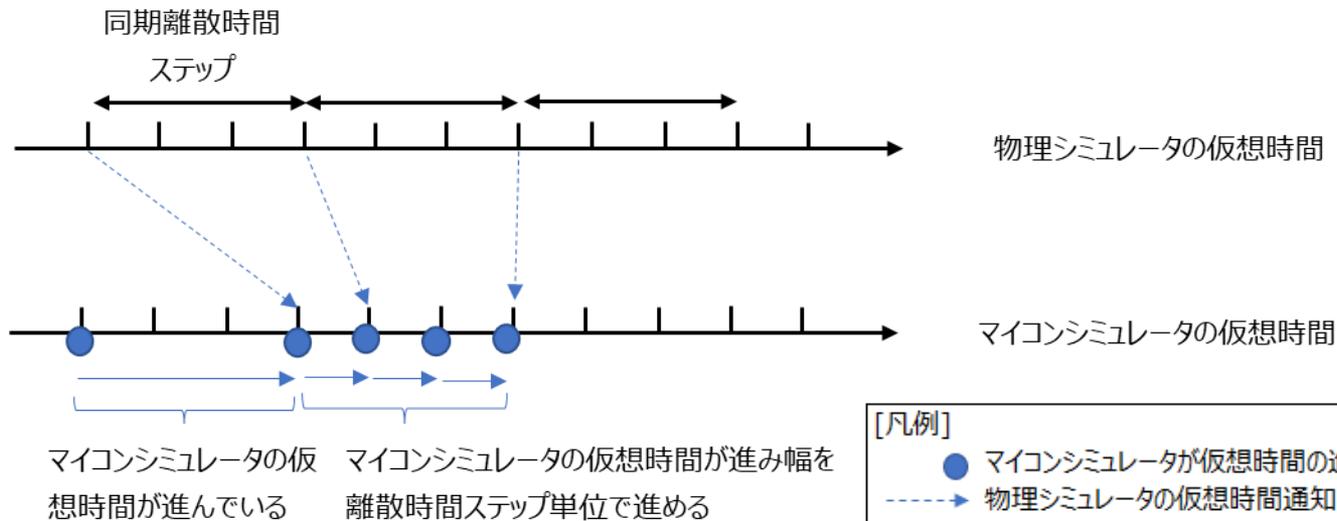
- 物理シミュレータから同期離散時間ステップ周期で、物理シミュレータの仮想時間を全マイコンシミュレータに通知する。

• 通信オーバーヘッドの削減手法

- 本通知の通信量は、シミュレーション実行速度に影響を与えるため、通信量削減が重要となる。そのための方法として以下の方法がある。
 - 物理シミュレータのセンシング情報を通知するタイミングで仮想時間を通知する。
 - 物理シミュレータとマイコンシミュレータが同一PC内に存在している場合は、メモリ共有することで通信量/遅延幅を低減させる。

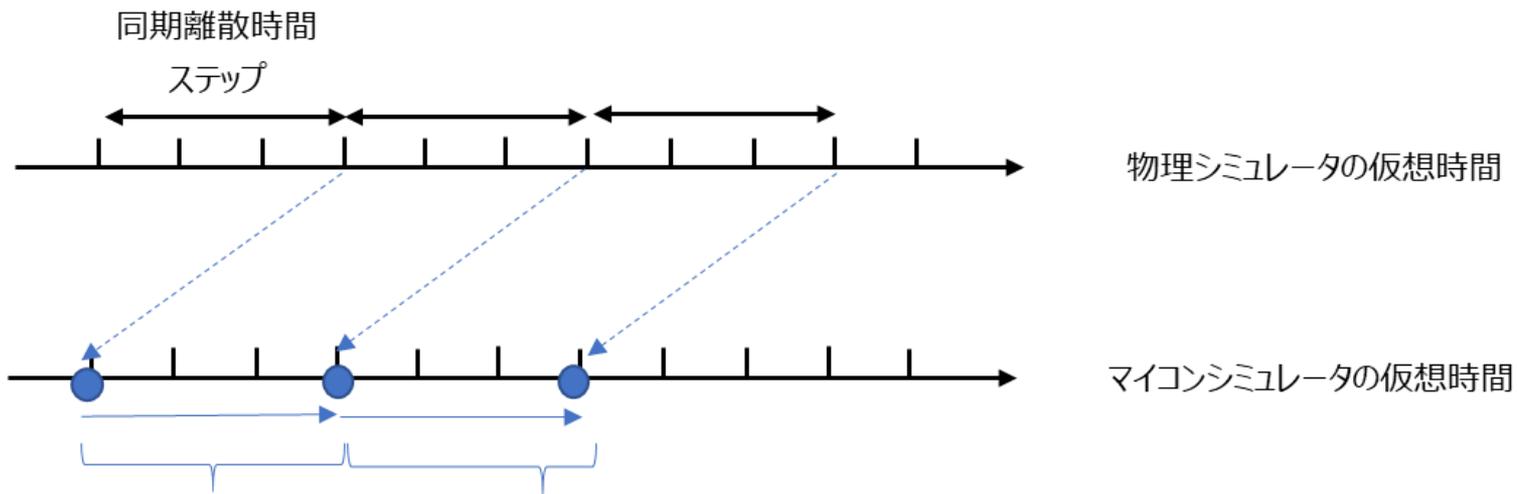
同期方法(マイコンシミュレータ)

- マイコンシミュレータの仮想時間と物理シミュレータの仮想時間を比較し、以下の方法で仮想時間を進める。
 - マイコンシミュレータの仮想時間が進んでいる場合
 - 仮想時間の進み幅を離散時間ステップ単位で進める(この間に、物理シミュレータの仮想時間が追いつくことを期待する)。



同期方法(マイコンシミュレータ)

- マイコンシミュレータの仮想時間が遅れている場合
 - マイコンシミュレータの仮想時間の進み幅を，物理シミュレータの仮想時間を超えない範囲で進める。

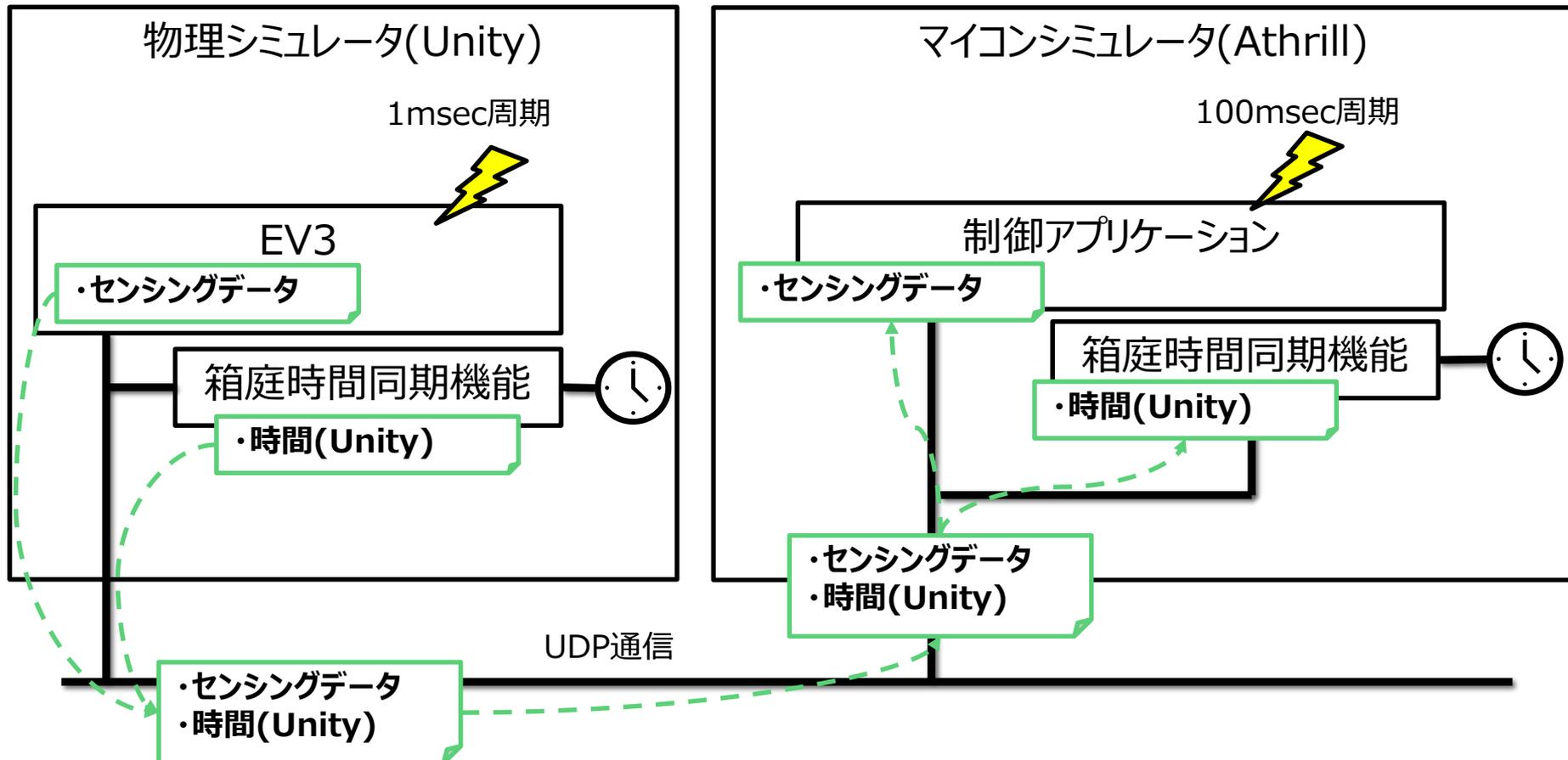


マイコンシミュレータの仮想時間が遅れているため，仮想時間の進み幅を大きく進める

以降，理想的には，マイコンシミュレータの仮想時間の進みは，物理シミュレータから見て，同期離散時間ステップだけ遅延する。ただし，各シミュレータの実行タイミング，通信遅延時間等によってその遅延幅はばらつく。

ETロボコン向けシミュレータへの適応方法

- 物理シミュレータのセンシングデータ送信タイミングで時間通知する



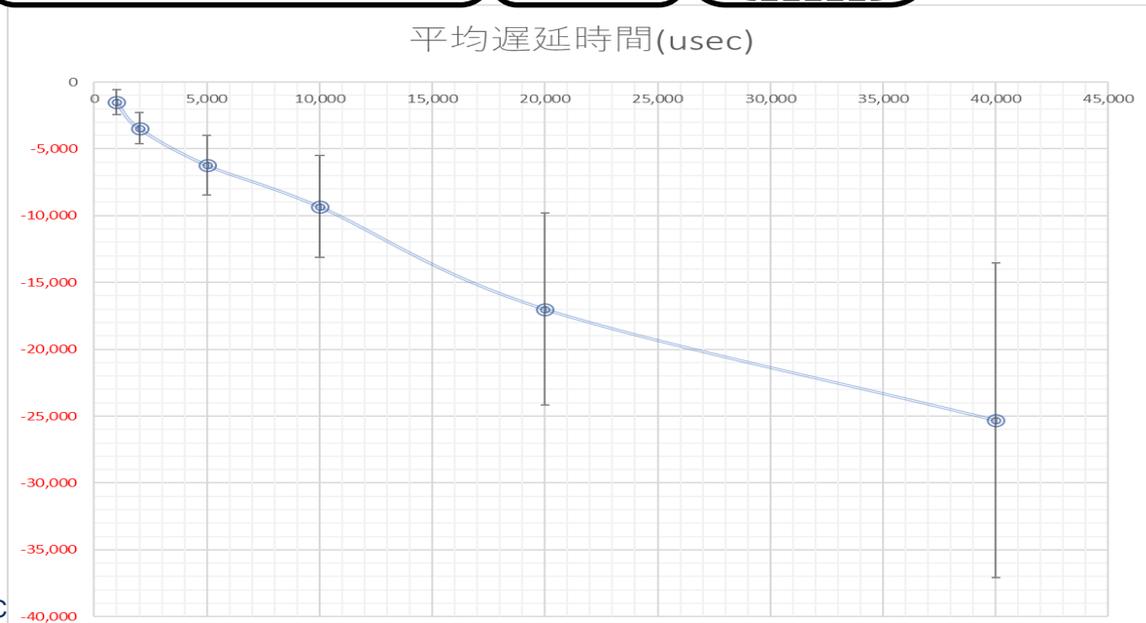
測定結果

- 同期離散時間ステップを40msec～1msecまで変化させて、遅延時間を計測しました
- シミュレーションとして、いずれの場合もライントレースは実行できました
- ※制御周期が100msecなので影響ないのかもしれませんが

計測結果サマリ

- 平均遅延時間はほぼ同期離散時間ステップ程度
- 同期離散時間ステップに比例してブレ幅は大きくなる
- 最大遅延幅は同期離散時間ステップ以上になる
- 最大遅延幅が収束しない原因は別途調査が必要
- ※汎用OSスケジューリングの限界？

同期離散時間(usec)	平均遅延時間(usec)	標準偏差(usec)	最大(usec)	最小(usec)
40,000	-25,328	11,766	-52,000	-1,000
20,000	-17,004	7,177	-37,000	-1,000
10,000	-9,330	3,819	-29,001	-1,000
5,000	-6,220	2,219	-24,001	-1,000
2,000	-3,460	1,174	-24,999	-999
1,000	-1,501	951	-24,081	-1,080



サマリ

• 今回試行したこと

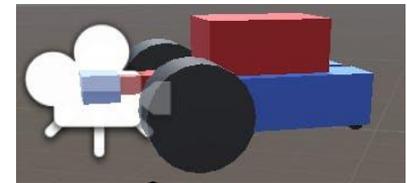
- シミュレーション時間同期の周辺概念を整理しました
- シミュレーション時間同期の方式を3パターン検討しました
- 1パターン(非完全同期(遅延幅観測可能))の考察を深めました
- ETOボコンシミュレータを利用して, 実現方法検討・実装・実験を実施しました.

• 気づいたこと

- シミュレーション時間同期方式の評価指標が必要
- 実物と比較して評価しないと何とも言えない

• 今後の課題

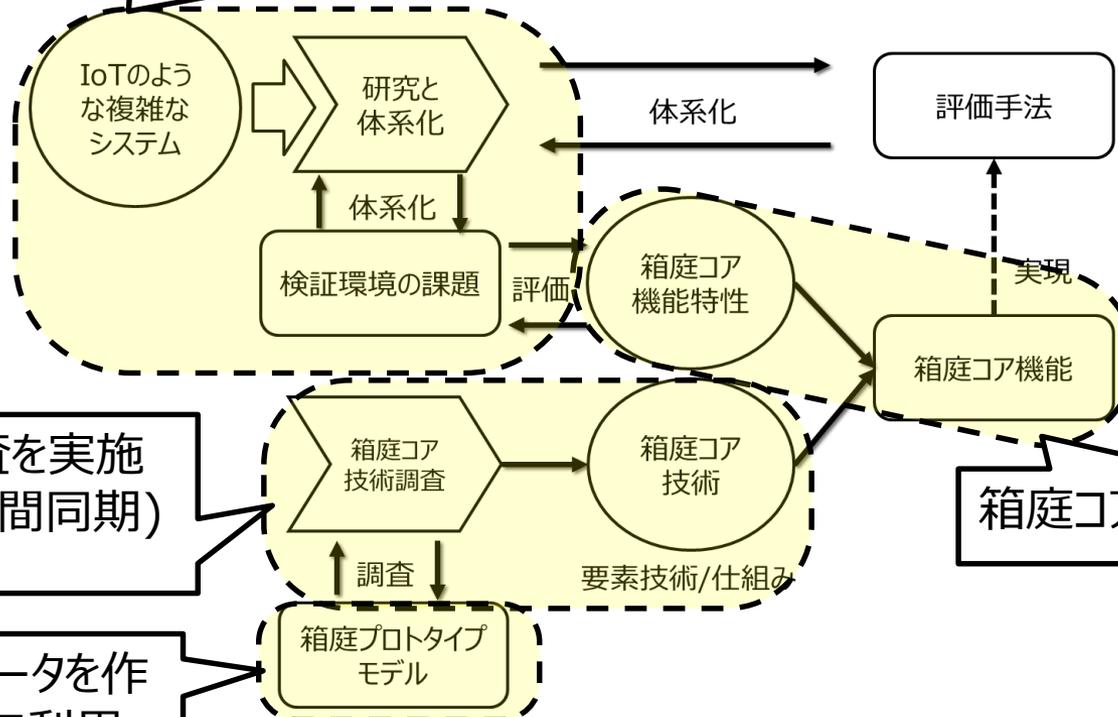
- 実物での評価結果との比較
- FMIの時間同期の仕様調査
- 協調シミュレーション精度の相場感を知る



全体サマリ

- 箱庭コア技術・機能を検討するためのプロセスを検討・整理し、全体的に一巡りしてみました。
- 箱庭WG結成してから約半年経ちましたが、少しずつ成果が出てきています。
(来年の活動をご期待ください！)

駆け足で検証環境の課題を分析・抽出



箱庭コア技術調査を実施
(シミュレーション時間同期)

ETロボコンシミュレータを作成・コア技術調査で利用

箱庭コア機能を整理