SDV 実現には「完全予定調和」からの超越が必要ではないか

AUTOMOTIVE WORLD 2025
SDV・電子開発コース トレンドセッション ①

宗像尚郎

ハイパフォーマンス コンピューティング プロダクトグループ
HPC SoC ソフトウエア イネーブルメント部
シニアダイレクタ

2025-1-22

自己紹介

ルネサスで R-Car SOC の SW マーケティング を行っています

- ルネサスエレクトロニクス株式会社 での活動
  - 最新の自動車向けの SW 要件（Requirement）収集 → 要件仕様書 の作成
  - クラウドや AI など広範な SW 技術トレンド理解 → 開発ロードマップ企画
  - 社内外 SW 開発部門に対するガイダンス → SW プロダクトの開発管理
  - ソリューション提案活動、差異化技術の紹介 → デバイスプロモーション活動
- オープンソース開発コミュニティ での活動（会社公認の社外活動）
  - AGL (Automotive Grade Linux) プロジェクト、yocto プロジェクト理事
  - COVESA（Connected Vehicle System Alliance）理事
  - SOAFEE、Jaspar、AUTOSAR などの 標準化団体とのコラボレーション推進
  - 各種 OSS プロジェクト、個人開発者との連携（コミュニティ開発リソースの活用）

自動車業界の「大規模 SW 開発へのさまざまな取り組み」を多面的に支援しています

自動車もまた「組み込み機器」であった

そもそも 組み込み (Embedded) とは？

非組み込み＝汎用（PC、スマフォ など）

- ユーザーによって用途が異なる
- 機器を購入したユーザーがアプリをインストールしてカスタマイズする
- さまざまな用途に対応できるよう計算資源には十分な余裕を持たせている
- メモリーやストレージの拡張が可能
- USB 等でユーザーが HW を拡張可能

組み込み（= 用途が特定されている）

- メーカーが特定用途向けに開発
- 同じ機能なので コスト競争が厳しい
- 故に、利用可能な 計算資源は最小限
- 原則として出荷後の SW 更新はない
- 専用オプション品以外の拡張性なし
- 代表例＝家電、産業機器、ゲーム機、ガラケー、自動車

特定用途向け「組み込み機器」の開発は、コスト競争から「制約条件との戦い」となる

組み込みソフトウエア の特性

組込み機器の制御には さまざまな制約 が伴う

- CPU の処理能力は限られている（時間制約）
  - 割り込みイベントに対する応答時間が規定されている（リアルタイム性）
  - 起動時間の制約（スタートアップ処理の軽量化）
  - 利用可能な電力の制約（発熱対策、電池駆動時間の確保など）
- 自由にメモリーサイズを拡張できない（容量制約）
  - RAM サイズ＝ワークメモリー、フレームバッファー（画面サイズ）
  - ROM サイズ＝プログラム容量の制約
- 無謬性（バグがひとつも無いことが求められる）
  - 原則、製品出荷後には SW を変更ができない（信頼性要求レベル は極めて高い）
  - ユースケースに基づいたシステム検証 によって十分なテストカバレージを確保

組み込み機器の制御用 SW では「多くの制約を回避する繊細な制御」を行っている

故に、組み込みソフトウエア開発者は「完全予定調和」を目指した

「完全予定調和（determinacy）」に求められる機能

- タスクの実行順序を完全に管理する → タスク優先度 による実行順序の制御
- 割り込み応答時間の保証 → 割り込み禁止区間 の設定による応答時間の管理
- メモリー管理 → 静的なメモリー割り付け
- HW 資源管理 → 静的な機能割り付け（特定アプリに HW 資源を占有させる）
- 障害解析、リカバリー → ログメッセージ埋め込み、エラーハンドル処理 の実装

- 組み込みシステムでは SW 設計者が「システム全体の挙動を完全に把握」し、想定されるあらゆる状況において システムが破綻しない ようにソフトウエアを設計し「完全な予定調和」 を実現させることに 全面的な責任を負っている

組み込み環境で HW/SW 構成が変化しなければ「完全予定調和」も達成可能だろう

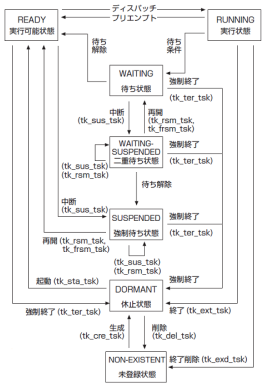
RTOS は組込み機器の制御用に開発された

RTOS は「完全予定調和」の基盤である

プログラマーがタスクの実行を完全にコントロールできる

- プリエンプション（preemption）
  - 優先度の高いタスクが 実行中のタスクに割り込む
  - プログラムからタスクを起床できるのでプログラマーの想定通り（deterministic）に 処理を記述できる
  - 割り込み処理ルーチンから直接タスクを起床できるので割り込み発生から実行までの時間を管理 できる
- プログラマーは 複数タスク間の状態遷移（＝予定調和）について全責任を持つ（マニュアル調停）
- 後でタスクを追加すると 予定調和が破綻 してしまう

図 1. タスク状態遷移図



T-kernel のタスクの状態遷移

https://www.tron.org/wp-content/themes/dp-magjam/pdf/t-kernel_2.0/html_ja/task_states_and_scheduling_rules.html

「RTOS」と「汎用 OS (Linux)」は根本的に違う

RTOS = 設計者が完全予定調和を作る

- RTOS の適用対象
  - 白物家電、ガラケー、産業機器
  - 自動車の ECU 制御用
- マイコン向け（基本はシングルコア）
- リアルタイム性が担保 される
- 機能安全 に対応した商用製品もある
- OSS と 商用製品 がある
  - OSS：FreeRTOS、Zephyr など
  - 商用：SafeRTOS、QNX など

Linux = OS の動的な自動調停に委ねる

- Linux の適用対象（非常に幅広い）
  - スーパーコンピュータ（科学、医療）
  - サーバー（金融、証券）
  - 車載（IVI）、Android スマートフォン
  - IoT、ルーター
- 1,000 コア、5,000 スレッドに対応済
- マルチユーザー、マルチタスク対応
- メモリー保護、仮想化 による分離
- オープンソース← 挙動解析が可能

RTOS と Linux では「タスクの実行順序決定のアルゴリズム」が大きく異なってくる

システム開発の新潮流：SW-First は破壊的（disruptive）である

古典的な 組み込み開発（例：ゲーム機）

- ハードウエア仕様が先 に決定される
- ベンダー固有機能 の利用も躊躇しない
  （非標準 API を持つアクセラレータ等）
- 毎回スクラッチ開発（再利用しない）
- 各チップ専用の開発環境
  - IP 専用の SW ライブラリー
  - BSP ← ボード専用の SW 環境
- ソフトには 完全予定調和 が求められる
  自動車では 機能安全 も要求される

SW First 指向（例：スマートフォン）

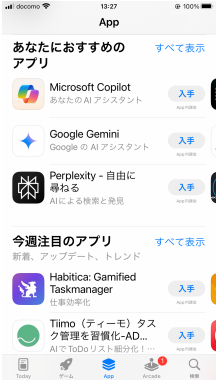
- 特定の HW に紐づいていない
- 自動調停で幅広い HW に適合できる
  - プロセッサー性能
  - メモリー、ストレージサイズ
  - 画面サイズ、横長／縦長
- アプリは複数が同時並行開発される
- HW と SW の進化は非同期となる
- アプリ間の実行調停は OS に委ねる
- 汎用高機能 OS 抜きには実現できない

この２つの考え方は根本的に異なっており、相容れない物であると認識する必要がある

スマフォでは 何故ユーザーによるアプリのカスタマイズができた？

スマートフォンの SW 更新の仕組み

- 全体更新（Firmware Over-the-Air; FOTA）
  - OS メジャーアップデート（アプリ更新も含まれる）
  - プリインストール SW の全体再構築（セキュリティ対策）
- 部分更新（アプリ単位の更新＝ SOTA）
  - App Store 経由でのアプリ単位で SW を更新
  - バージョンアップ、新規インストール、削除
  - ユーザー毎に SW 構成が異なるので、によるアプリ更新時にユーザー「全体結合検証」を行うことはできない
  - OS ベンダーが用意した「互換性テストプログラム」によるアプリ単体検証 でシステム整合性を担保している



スマートフォンは全体結合検証で「予定調和」を担保するという戦略は採用していない

スマートフォン は「組み込み機器」ではない

スマートフォンは「汎用 OS の自動調停機能」に 100% 依存している

- 「ユーザーによるアプリインストール」、「SW の逐次更新」はスマートフォンの本質なので、開発者による SW の予定調和は意図的に断念した。その代わりに「汎用 OS 内蔵の自動調停機能」に委ねることで「SW First の世界観」を実現

- 「汎用 OS 内蔵の自動調停機能」
  - タスク スケジューリングの動的な自動最適化機構
  - 割り込み処理の最適化
  - 投機的なデータキャッシュ（空きメモリーはソフトキャッシュ機構に全面活用）
  - リソースのアロケーション、プロセス間共有（メモリー、HW 資源）
  - システム プロファイリング、デバッグ支援機能

スマフォの信頼性担保の拠り所は、組み込み機器の「ガラケー」とは全く異なっている

精緻な自動車の制御を「汎用 OS の自動調停」に委ねられるか？

自動車もまた「組み込み機器」であった　← その本質は何も変わっていないが...

- 依然として「組み込み」な部分（固い部分）
  - 走る・曲がる・止まる
  - エンジン点火制御／バッテリーマネージメント
  - レガシーなボディ・シャーシ—コントロール
  - 運転支援（ADAS）／自動運転（AD）
- スマフォライクなエキスペリエンスが期待される部分（やわらかい部分）
  - インフォテイメント（マルチメディア、コネクティビティ）
  - アプリストアを使ったカスタマイズへの対応
  - クラウドサービス連携（AI 対応、エージェント対応、IT 連携）

クルマ全体をスマフォのように扱えないので、パーティショニングが必要となる

「パーティショニング」に社内政治を持ち込まない

守旧派（「組み込み」苦労人）

- 既存コードには手を触れさせたくない
  - リアルタイム命
  - 触らぬコードに ····
- 汎用 OS を理解しようとしない
  - 投機的なスケジューリング機構
  - 動的メモリー管理機構
- RTOS 上に Linux 互換タスクを追加
  - POSIX 対応 API だけを提供する
  - 汎用 OS の調停機構は利用しない

革新派（IT/Cloud テクノロジー気触れ）

- 全面的に「スマフォ流」を推奨
  - コードサイズの肥大化
  - システム起動時間が長大
  - システム検証時間を確保しない
- 自動車特有の要件を理解しない
  - ネットワーク常時接続が前提
  - 計算資源に十分な余力があると想定
  - 厳しいコスト制約を理解できない
  - レガシーな技術を取り込めない

自動車業界では「組み込み」派が主流だが、新潮流も取り込まないと流れに乗れない

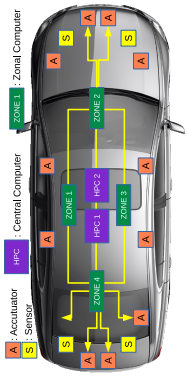
SDV では汎用 OS とも向き合わねばならない

SDV には汎用 OS を実行可能な Vehicle Computer が搭載される

SW の部分更新には汎用 OS による自動調停が必須である

- インフォテイメント（HMI）
- ゲートウエイ（ネットワーク ルーター）

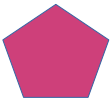
- Vehicle Computer（HPC）というアーキテクチャ革命
- HPC は専用コンピュータ（＝組み込み機器）ではない
- さまざまなワークロードへの対応 が可能になる
  - 従来、専用 ECU で実現していた 複数の機能の集約
  - リアルタイム性の保証が必要なユースケース もある
  - 機能安全要件 への対応が必要な部分も出てくる
  - クラウド連携 や サイバーセキュリティ対策 への考慮も必要



従来専用 ECU で制御していた処理の一部を HPC へ集約し、機能更新を可能にしたい

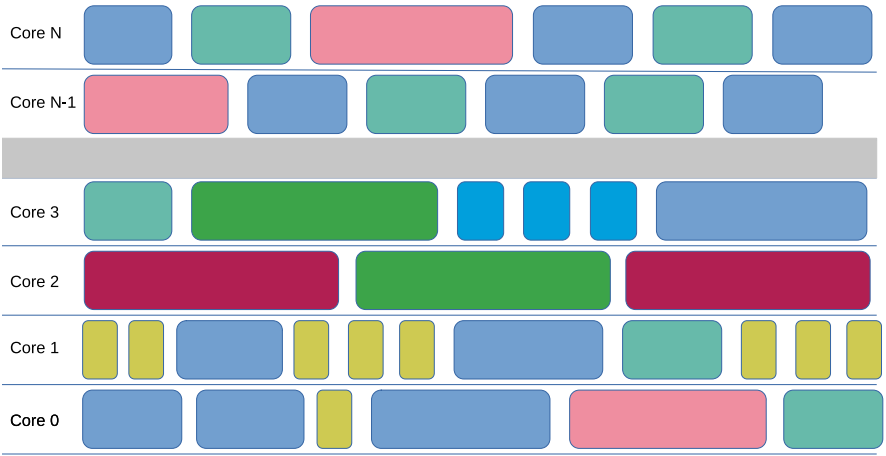
RTOS のプログラムは、それぞれがローカルに最適化されている

- 個々の RTOS プログラムは 個別に部分最適化されている（予定調和）
  - 静的なメモリー割り付け
  - 厳密なシーケンサー動作（リアルタイム）
  - ローカルなインターフェース（互換性なし）
- 以下の発想では設計されていない
  - 互換性のある API の利用
  - 共通デザイン
  - システム全体での性能最適化

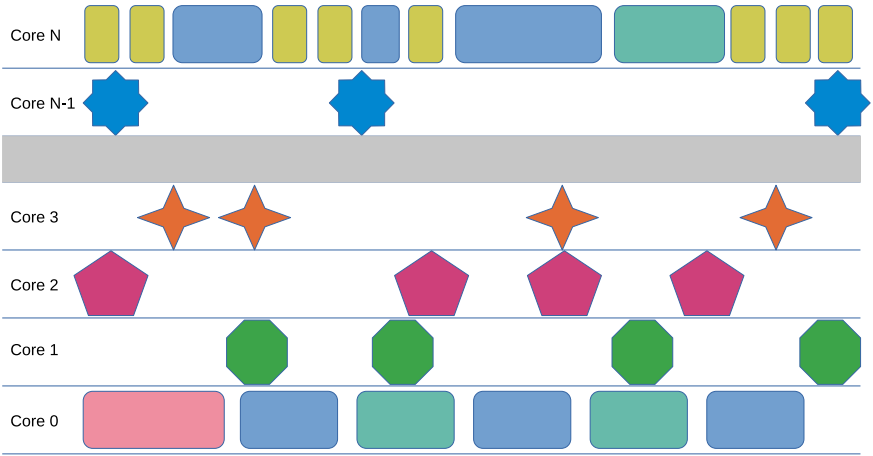


既にローカルに個別最適された RTOS のプログラムには、誰も手をいれたがらない

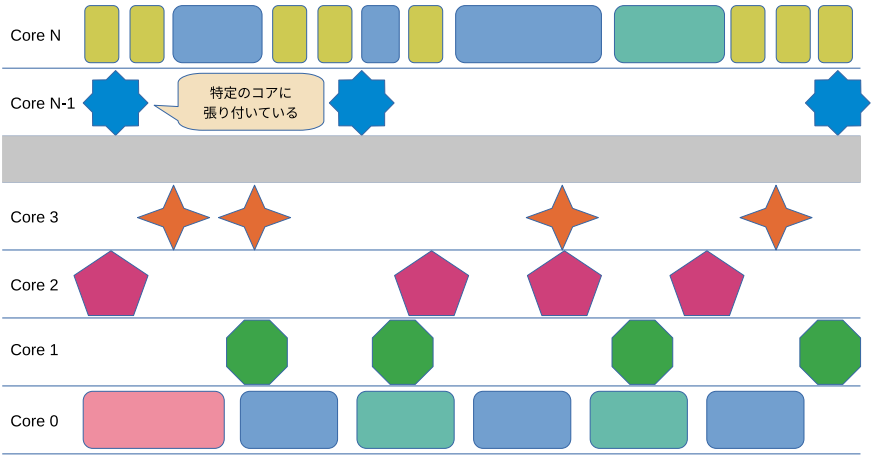
Linux では動的なスレッド実行調停によって性能が最適化されている



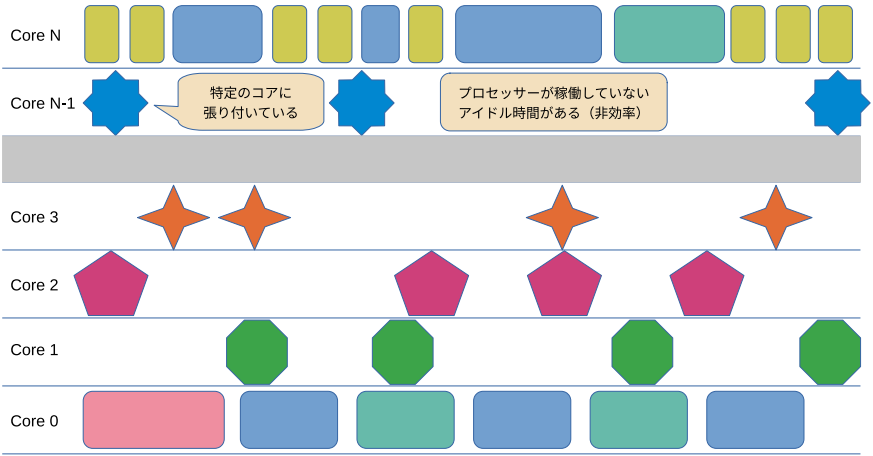
既存の RTOS プログラムを単純に HPC に移植すると...



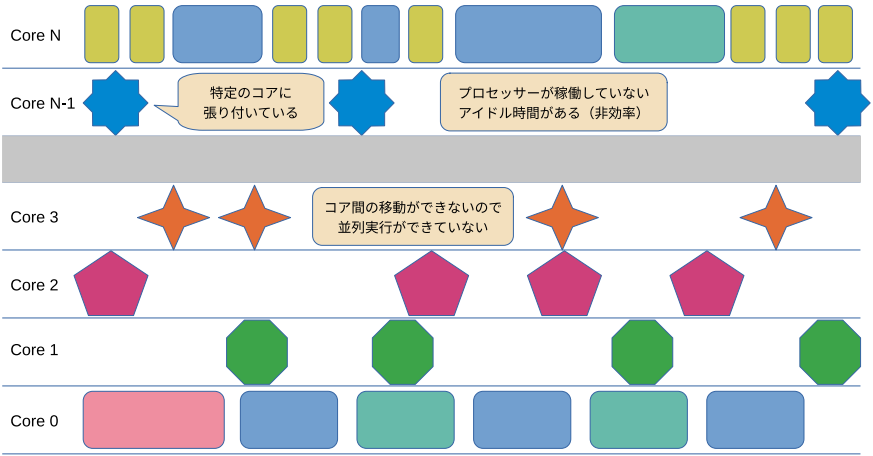
既存の RTOS プログラムを単純に HPC に移植すると...



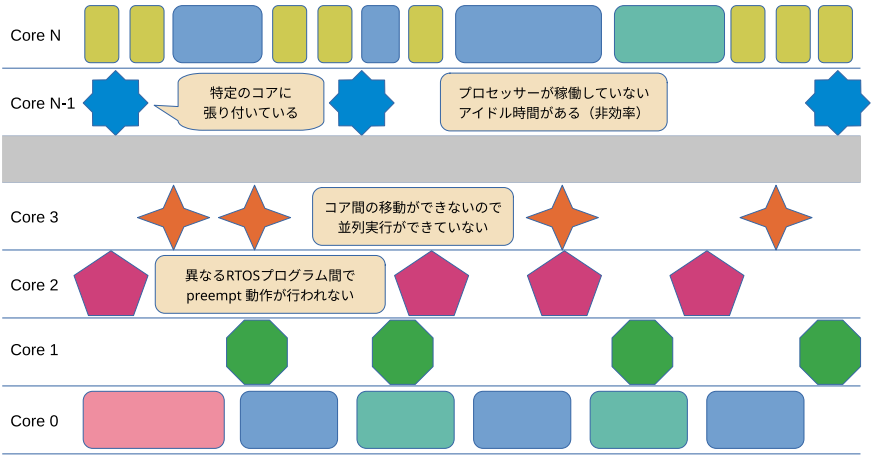
既存の RTOS プログラムを単純に HPC に移植すると...



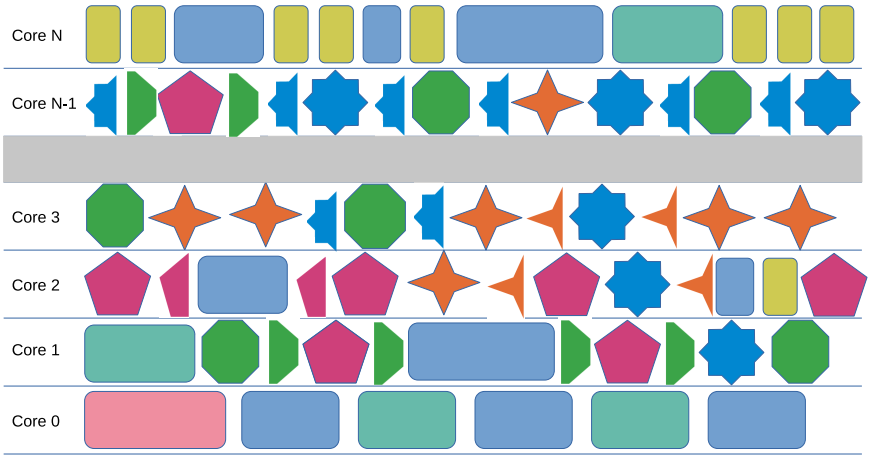
既存の RTOS プログラムを単純に HPC に移植すると...



既存の RTOS プログラムを単純に HPC に移植すると...



HPC 上での RTOS プログラムの理想的な動作 （これは簡単ではない）



HPC を生かすための SW の移植戦略

マルチコアを活かすには 処理の密度ではなく並列度を高める戦略 が重要

- RTOS の処理内容を精査する
  - デッドライン要件 の精査（そのまま移植しない）
  - プログラム内で CPU 開放 ができる処理がないか
- 処理の 同期ポイント（待ち合わせ）が最少 となるステートレスな構造に見直す
- OS の自動調停機能 を有効活用する
  - リアルタイム タスクの利用を最少限に する
  - スケジューリング パラメータ を最適設定する
  - デッドラインのある処理は コプロセッサー等にオフロード する
  - 充分な メモリーリソース をアロケートする
- ハイパーバイザー による HW 分割（ゲスト OS 化、監査機能の利用）
- コンテナ によるアプリケーションのカプセル化

まとめ

- 特定用途向けの組み込み機器 では、コスト競争のための 必要最小限のリソース という制約下で設計者が RTOS を使って「完全予定調和」の世界観を構築する。
- パソコンやスマートフォンでは完全予定調和は作れないので、余裕のあるリソース上で 汎用 OS の「動的自動調停機構」をつかってシステムを動かしている。
- SDV は能力に十分余裕がある Vehicle Compute（HPC）上に「動的自動調停」ができる高機能 OS を搭載できること。製品出荷後の SW 部分更新 によって ユーザーに最先端のユーザー体験（価値）を提供することができる。
- 「完全予定調和」が作りこまれた RTOS の SW を HPC にそのまま移植 すると（= Linux を RTOS のように構成すると）様々な弊害 がある。HPC の特性である処理の並列度 を最大限に発揮できるＳＷ構造を検討することが 成功への指針 となる。