ETロボコン向け TOPPERS活用セミナー

ソフトウェアプラットフォームとRTOSの基礎

2017年6月17日

高田 広章

NPO法人 TOPPERSプロジェクト 会長
名古屋大学 未来社会創造機構 教授
名古屋大学 大学院情報学研究科 教授
附属組込みシステム研究センター長
Email: hiro@ertl.jp　URL: http://www.ertl.jp/~hiro/

目次

TOPPERSプロジェクトの概要

TOPPERSプロジェクトとは?

- ITRON仕様の技術開発成果を出発点として，組込みシステム構築の基盤となる各種の高品質なオープンソースソフトウェアを開発するとともに，その利用技術を提供

組込みシステム分野において，*Linux*のように広く使われるオープンソース*OS*の構築を目指す!

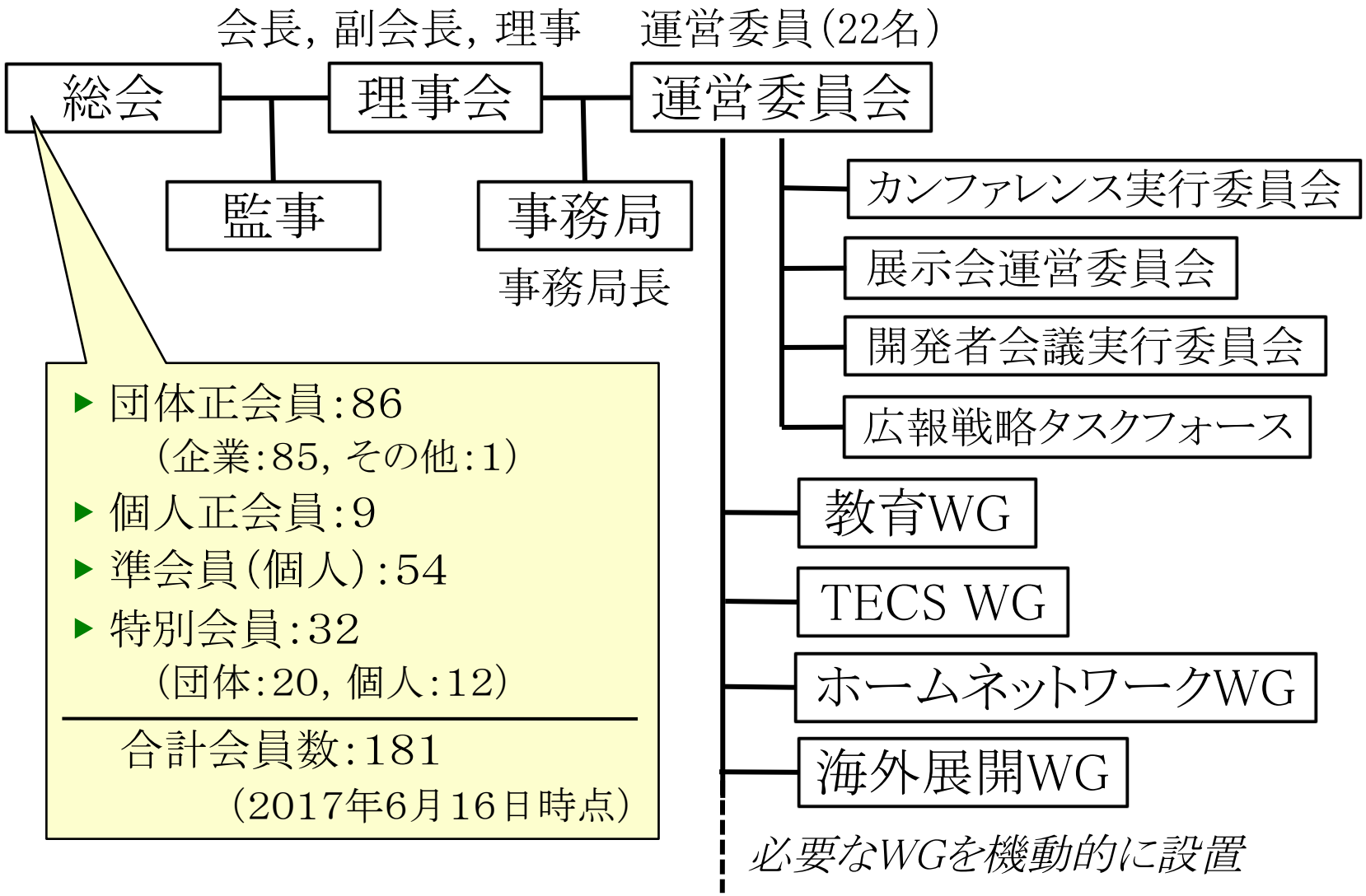
プロジェクトの狙い

- 決定版のITRON仕様OSの開発 ← ほぼ完了
- 次世代のリアルタイムOS技術の開発
- 組込みシステム開発技術と開発支援ツールの開発
- 組込みシステム技術者の育成への貢献

プロジェクトの推進主体

- 産学官の団体と個人が参加する産学官民連携プロジェクト
- 2003年9月にNPO法人として組織化

NPO法人 TOPPERSプロジェクトの会員と組織

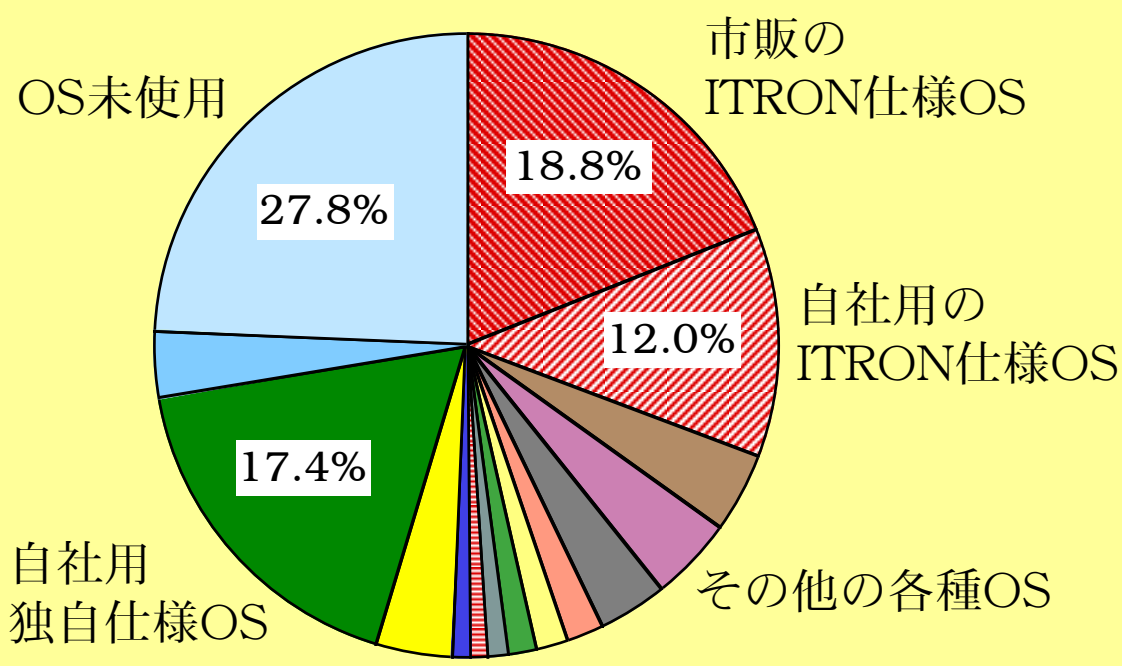


ITRON仕様とは

- トロンプロジェクトにおいて標準化してきた組込みシステム向けのリアルタイムカーネルとそれに関連する仕様
- 誰でも自由に実装できるオープンな仕様

! それに準拠して実装されたOSはオープンとは限らない

- 産学共同の標準化活動の成果
- 多くのプロセッサ上に実装され，国内では業界標準に



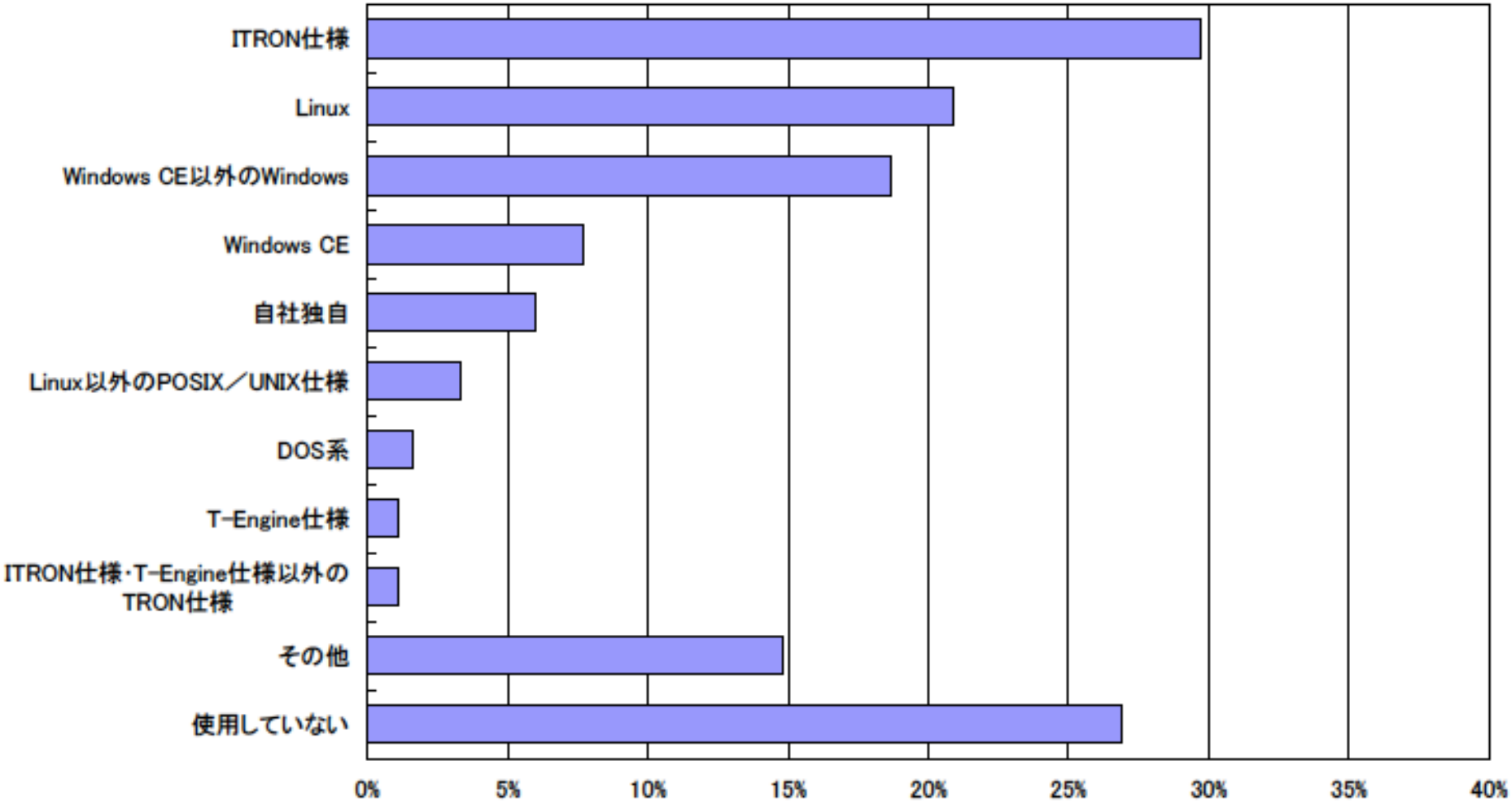
最近開発した組込み機器に使用したOS

（トロン協会による調査，1998年末～1999年頭，日本）

Q2-7 ターゲットOSの種類

複数選択

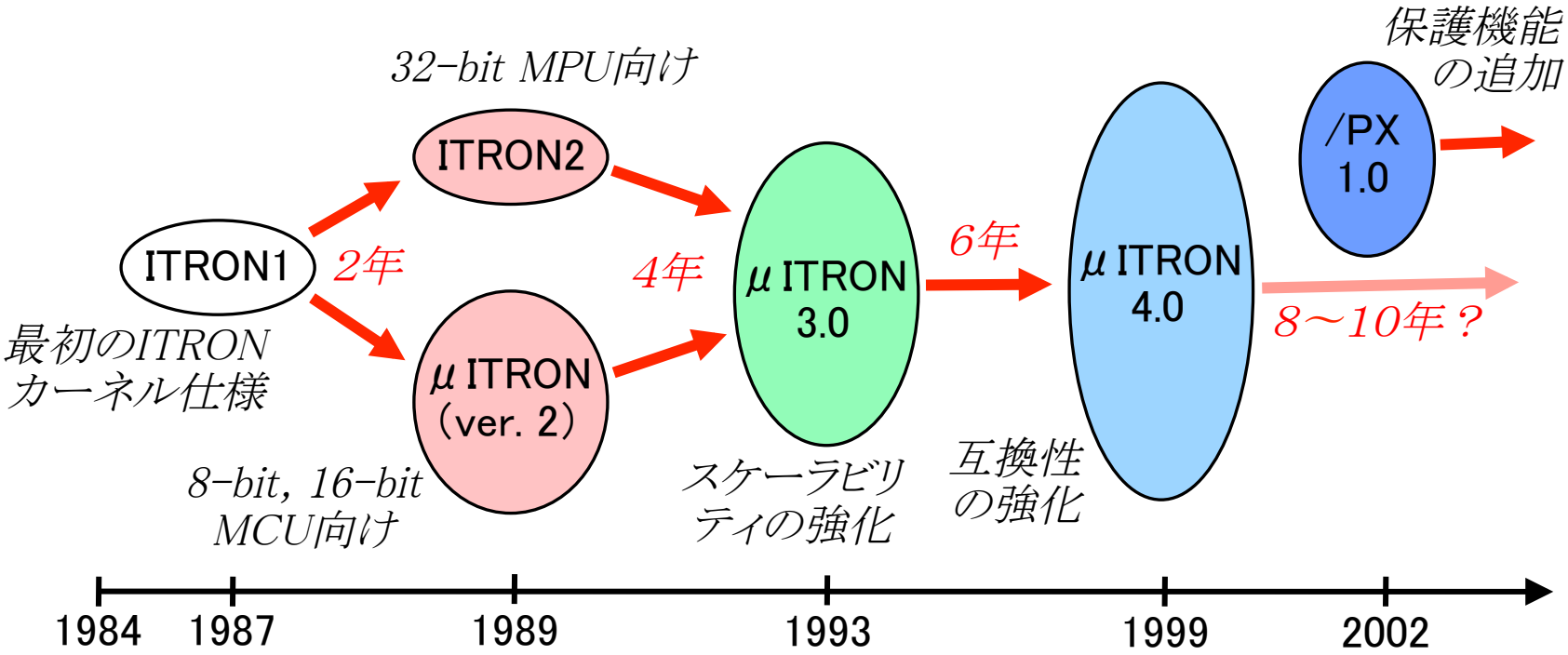
プロジェクト責任者



使用プロジェクトの割合

ITRONカーネル仕様の歴史

- プロジェクト開始から約20年間に，4世代のITRONカーネル仕様を策定・公開
- μITRON4.0仕様は，この世代のリアルタイムOS技術の範囲では，極めて完成度の高い仕様に



ITRONカーネル仕様の特徴

- OSの小型軽量化が可能
  - ワンチップマイコンにも適用可能
- 仕様の理解が容易
  - ! 技術者教育のための標準化の側面を重視
- 完全にオープンな標準仕様
  - ロイヤリティなしで実装することができる
- 多種多様なプロセッサ用に実装できる/されている
  - 8-bitワンチップマイコンから64-bitマイコンまで
  - 異なるプロセッサへの移行が容易に
- 多くの機器で使用実績がある
  - 組込みシステム分野で最も広く使われているOS仕様
- 多くのメーカ/ベンダがサポート

ITRON仕様カーネルの開発状況

- 多くのITRON仕様準拠のRTOSが開発・販売された
- 自社用に実装した例も多数
- いくつかのオープンソースの実装

ITRON仕様カーネルの主な適用機器例

AV機器，家電，個人用情報機器，娯楽/教育機器

- テレビ，ビデオ，デジタルカメラ，STB，オーディオ，電子レンジ，
- 炊飯器，PDA，電子手帳，カーナビ，ゲーム機，電子楽器

パソコン周辺機器/OA機器

- プリンタ，スキャナ，ディスクドライブ，DVDドライブ，コピー，FAX

通信機器

- 留守番電話機，携帯電話機，放送機器，無線設備，人工衛星

運輸機器，工業制御/FA機器/設備機器，その他

- 自動車，プラント制御，工業用ロボット，自動販売機，医療用機器

TOPPERSの主な開発成果物

一般公開しているもの

第1世代カーネル

- TOPPERS/JSPカーネル，TOPPERS/FI4カーネル
- TOPPERS/ATK1（Automotiveカーネル バージョン1）
- TOPPERS/FDMPカーネル，TOPPERS/HRPカーネル

新世代（第2世代）カーネル

- TOPPERS/ASPカーネル，TOPPERS/SSPカーネル
- TOPPERS/FMPカーネル，TOPPERS/HRP2カーネル

第3世代カーネル（ITRON系）

- TOPPERS/ASP3カーネル

AUTOSAR関連

- TOPPERS/ATK2（Automotiveカーネル バージョン2）
- TOPPERS/A-COMSTACK，TOPPERS/A-WDGSTACK
- TOPPERS/A-RTEGEN

ミドルウェア

- TINET，FatFs for TOPPERS
- CAN/LINミドルウェアパッケージ
- TOPPERS/ECNL（ECHONET Lite通信ミドルウェア）
- RLL（Remote Link Loader），DLM（Dynamic Loading Manager）

ツール，その他

- TECS（TOPPERS組込みコンポーネントシステム）
- SafeG（高信頼組込みシステム向けデュアルOSモニタ）
- EV3RT（LEGO Mindstorms EV3向けSPF）
- TLV（TraceLogVisualizer），TOPPERS Builder

教育コンテンツ

- 基礎1・基礎2・基礎3実装セミナー教材
- 基礎ハードウェア設計セミナー教材
- ETロボコン向けTOPPERS活用セミナー教材

TOPPERSライセンス

- TOPPERSプロジェクトで独自に開発したソフトウェアには，独自のライセンス条件を設定する

基本的な考え方

- 組込みシステムの事情を考慮し，GNU GPLやBSDライセンスより自由に使えるライセンス条件とする
- 成果をアピールすることが開発資金獲得に繋がることから，どこでどう使われているかをなるべく知りたい

ライセンスの内容

- 派生物をオープンする義務は課さない．派生物を販売するビジネスも可能
- 機器に組み込んで使用する場合の実質的な義務は，利用したことを報告することのみ … **レポートウェア**

開発成果物の主な利用事例

エスクード（スズキ）

スカイラインハイブリッド（日産）

IPSiO GX e3300（リコー）

H-IIB（JAXA）

Cell³iMager duos
（SCREEN
ホールディングス）

OSP-P300
（オークマ）

SoftBank
945SH
（シャープ）

UA-101（Roland）

PM-A970(エプソン)

次の10年を見据えた活動指針 (2011年度に策定)

Smart Futureのための組込みシステム技術

- 組込みシステム技術を，持続可能なスマート社会の実現（Smart Future）のための重要な要素技術の1つと位置づけ，その研究開発と普及に取り組む
- それに向けての研究開発課題
  - Safety & Security
  - Ecology（高エネルギー効率）
  - Connectivity

コンソーシアムによるオープンソースソフトウェア開発

- 同じ技術に関心を持つプロジェクトメンバによりコンソーシアムを結成し，複数組織の協力によりソフトウェアを開発
- 開発したソフトウェアは，TOPPERSプロジェクトからオープンソースソフトウェアとして公開

重点的に取り組んでいるテーマ

※ SPF：ソフトウェア
プラットフォーム

次世代のリアルタイムカーネル技術

- TOPPERS第3世代カーネル（ITRON系）
- 車載システム向けRTOS（AUTOSAR OS仕様からの発展）

ソフトウェア部品化技術

- TECS（TOPPERS組込みコンポーネントシステム）

組込みシステム向けSPFと開発支援ツール

- 車載制御システム向けSPF（AUTOSAR仕様ベース）
- 仮想化技術（SafeG，A-SafeG），ホームネットワーク
- 宇宙機向けSPF（SpaceWire OS）
- 開発支援ツール（シミュレータ，可視化ツール）

技術者育成のための教材開発

- プラットフォーム技術者の育成
- ETロボコン向けSPFと教材の提供

リアルタイムカーネル開発の流れ

! 高信頼性・安全性・リアルタイム性を追求

第１世代のリアルタイムカーネル

- μITRON4.0仕様準拠＋αのリアルタイムカーネル
  - TOPPERS/JSP, FI4, FDMP, HRP
- OSEK/VDX OS仕様準拠のリアルタイムカーネル
  - TOPPERS/ATK1

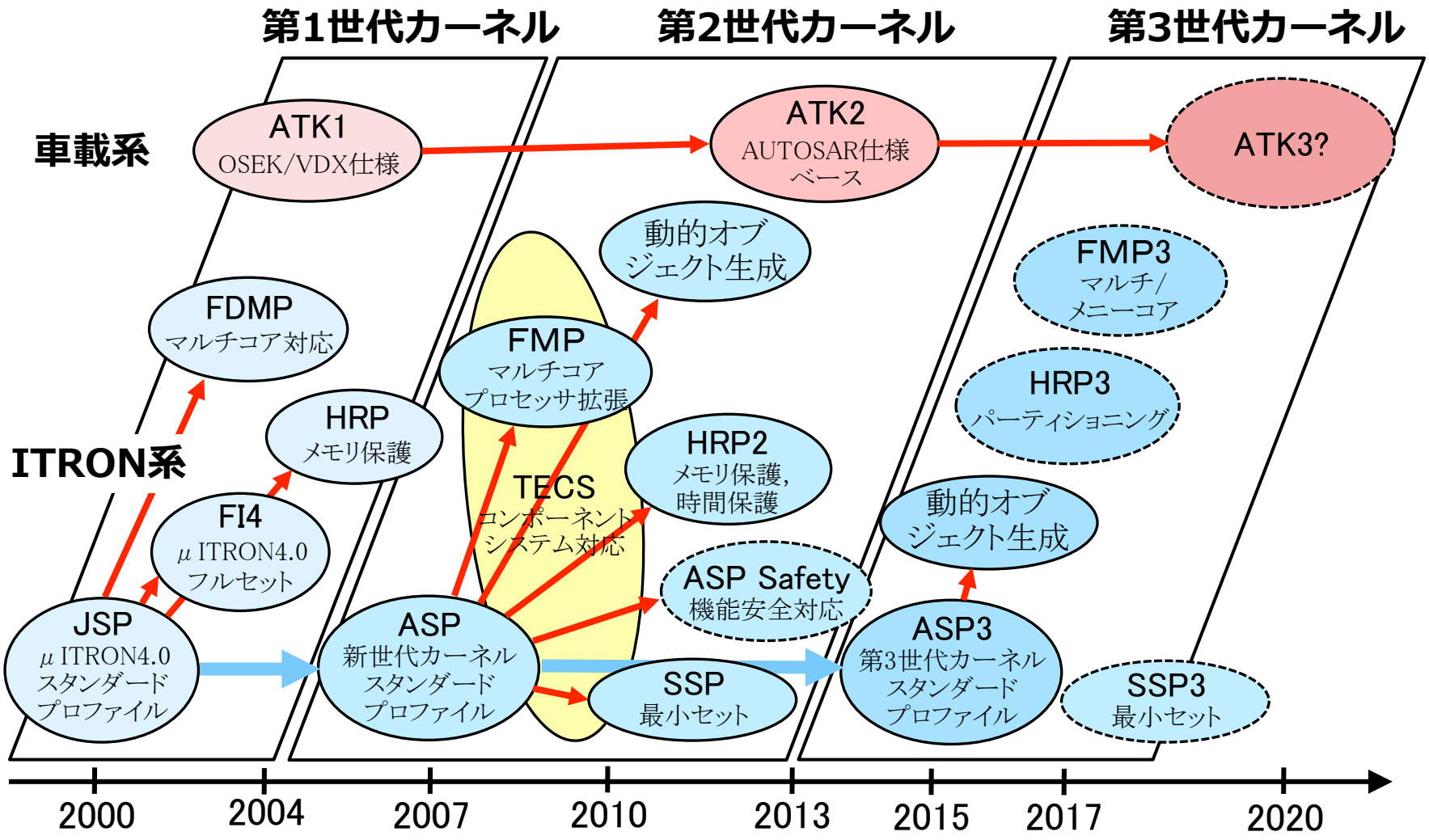
第２世代のリアルタイムカーネル

- TOPPERS新世代カーネル（ITRON仕様からの発展）
  - TOPPERS/ASP, FMP, HRP2, SSP
- AUTOSAR OS仕様ベースのリアルタイムカーネル
  - TOPPERS/ATK2（SC1, SC3, SC1-MC, SC3-MC, …）

第３世代のリアルタイムカーネル

- TOPPERS第3世代カーネル（ITRON系）
  - TOPPERS/ASP3, …

TOPPERSカーネル開発ロードマップ



ソフトウェアプラットフォームとその必要性

組込みシステム開発を取り巻く状況

半導体技術，ネットワークの進歩

組込みシステムの大規模化・複雑化

- 機器の複合化・デジタル化・ネットワーク化
- 制御要素に情報処理要素が複合
- コンピュータ制御による高機能化・高付加価値化

ネットワーク接続とクラウド連携の拡大

- 組込みシステムと情報システムを結合した大規模なシステム（統合システム，CPS，IoT）の構築が重要に

組込みシステムの適用分野が拡大

- コンピュータの小型化・低価格化により広がる適用分野

開発期間の短縮やコストダウンに対する要求

- 新興国との競争の中で今まで以上のコストダウン要求

（単一の）プロセッサの高速化の限界

- 消費電力（=発熱量）が最大の制約条件に

組込みシステムの開発効率化と品質確保のために

すぐにできること

- ソフトウェア開発プロセスの地道な改善
  - CMMI, SPICE, 機能安全規格等が参考に
  - 取り組みにあたっては，本質を見失わないことが重要

! 人材育成 … 時間はかかるが着手はすぐにできる

中期的に取り組むべきこと

- 設計抽象度を上げる＆設計の早い段階での検証
  - モデルベース／モデル駆動開発の流れ
  - 仮想環境（シミュレータ）によるシステム検証
- 設計資産の再利用を促進する仕組みの構築
  - 差分開発が中心の組込みシステム開発では特に重要
- 応用分野毎のプラットフォームの構築・活用と共通化
  - プラットフォーム化による再利用性と品質の向上

プラットフォームの構築・活用と標準化

プラットフォームとは？

- プラットフォーム＝ハードウェアPF＋ソフトウェアPF
- ソフトウェアPF（SPF）＝OS＋ミドルウェア＋デバドラ

応用ドメイン毎のプラットフォーム構築と活用

- 適切な範囲（これが難しい）の応用ドメインを設定し，それに向いたプラットフォームを構築
- 構築したプラットフォームを，多くのアプリケーションに活用

プラットフォーム化による再利用性と品質の向上

- アプリケーションの再利用性が向上することに加え，プラットフォーム自身の再利用による開発コスト減も
- システムの品質確保の鍵となる部分であり，高品質なプラットフォームの構築は，システムの品質向上につながる
- 新機能/新技術の導入が容易に

プラットフォームの標準化の意義

- プラットフォームは，業界内での標準化によりその導入意義が増す
- プラットフォームの独占/寡占の回避
  - 特定ベンダのプラットフォーム製品に縛られることを避けることは，コスト最適化の上で重要
  - プラットフォーム（特にSPF）は，独占/寡占が起こりやすい分野
  - プラットフォームの（実質的な）独占/寡占は，独占/寡占したベンダの力を極めて強くする
  - 自動車業界がリソースをかけて標準化を推進する理由
- ソフトウェア構築手順の標準化
  - 自動化（ツール化）が促進される
  - 分散開発が容易に
- 技術者の確保/育成

プラットフォームの共通化・標準化の例

- 社内でのプラットフォーム共通化

  例）パナソニックのUniPhier（ユニフィエ）

  - プロセッサとビデオコーデックなどを含むシステムLSIと，ミドルウェアやOSなどのソフトウェアからなるデジタル家電用の統合プラットフォーム
- 業界内でのプラットフォーム標準化
  - 自動車制御システム向けのプラットフォームやツールの標準化
    - AUTOSAR（http://www.autosar.org/）
    - JASPAR（http://www.jaspar.jp/）
- デファクト標準によるプラットフォーム標準化
  - スマートフォン向けのプラットフォームは2つに収束している（3つめがあるかも？）

AUTOSAR (AUTomotive Open System ARchitecture)

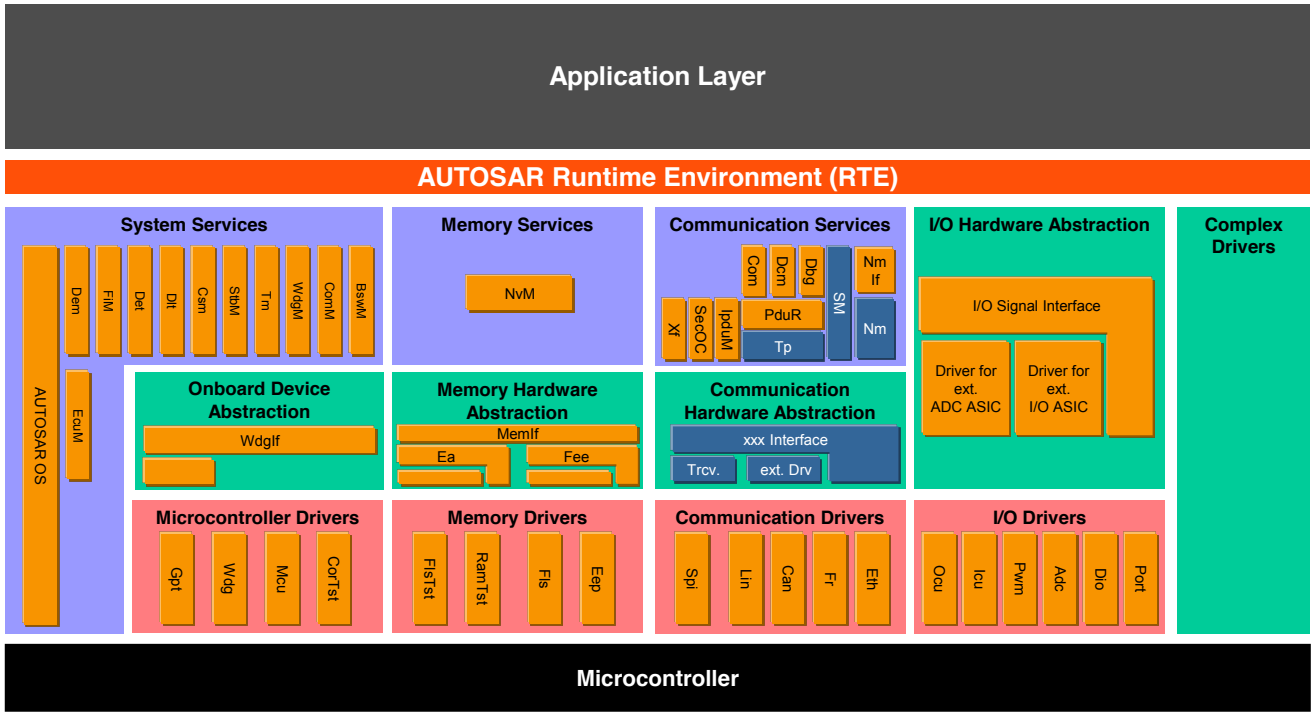
- 自動車，自動車部品，エレクトロニクス，半導体，ソフトウェア企業によるグローバルパートナーシップ（2003年に設立）
  - ソフトウェアの複雑性を軽減するために，ソフトウェア基盤（infrastructure）の業界標準を作成
- コアパートナー（2017年時点）

BMW	Daimler	PSA Peugeot Citroen
Bosch	Ford	トヨタ自動車
Continental	GM	Volkswagen

- 最新の仕様書（群）はRelease 4.3.0
  - Release 4.2.2の時点で，約100のソフトウェア仕様書と約110の関連ドキュメント，その他多くのファイルが含まれており，全体で19600ページ（と言われている）
- “Cooperate on standards, compete on implementation”（標準化で協調し，実装で競争する）をポリシーに

AUTOSARソフトウェアプラットフォームの構成

- Runtime Environment (RTE)
  - SW-C間，SW-CとBSW間の通信インタフェースを提供
- Basic Software（BSW）
  - OS+デバイスドライバ+ミドルウェア群



AUTOSAR Layered Software Architecture 4.2.2より

RTOS入門

リアルタイムシステムとは?

JISによる「実時間処理」の定義

- 計算機外部の処理に関係を持ちながら，かつ外部の処理によって定められる時間要件にしたがって，計算機の行なうデータ処理

リアルタイムシステム研究者の間で一般的な定義

- 処理結果の正しさが，出力される結果値の正しさに加えて，結果を出す時刻にも依存するようなシステム

! 単に速い応答を求められるシステムをリアルタイムシステムと呼ぶわけではない

多くの組込みシステムはリアルタイムシステム

- 組込みシステムは，機械・機器を制御するシステムであり，制御対象の時間要件にしたがうことが必要
- 一部の処理のみにリアルタイム性が求められる場合も

RTOS (リアルタイムOS) とは?

- 文字通り, リアルタイムシステム構築のためのOS
- 具体的には, 次のような特徴を持つOS(これらの特徴をすべて持っているとは限らない)

(1) リアルタイムシステム向けの機能を持つ
  - プリエンプティブな優先度ベーススケジューリング
  - 優先度継承や優先度上限プロトコルのサポートなど

(2) 予測可能性を持つ
  - OSの各サービス時間があらかじめわかっている
  - ただし, 予測可能性にもいろいろなレベルがある

(3) 時間制約を管理 ← 一部の研究ベースのRTOS
  - OSが各処理の時間制約を考慮してスケジューリング

(4) 高速に応答(制御対象に対して十分に)

OSの機能面から見た組込みシステムの特性

OSが必ずサポートしなければならないI/O装置はない

- 多くの組込みシステムが共通に持つI/O装置が無い
  例）ストレージデバイスを持たない組込みシステムも多い
- 複数のアプリケーションで同一のI/O装置を共有する状況は少ない

保護のための機能は必須ではない

- 組込みソフトウェアは，組込み対象の機器を制御することのみを目的に設計される
- 組込みソフトウェアは，機器に固定されている

→ デバッグが終われば，アプリケーションソフトウェアは信頼できるという前提が成り立つ

! 汎用システムと近い性質を持つ組込みシステムも多くなっている（典型例は，スマートフォンやカーナビ）

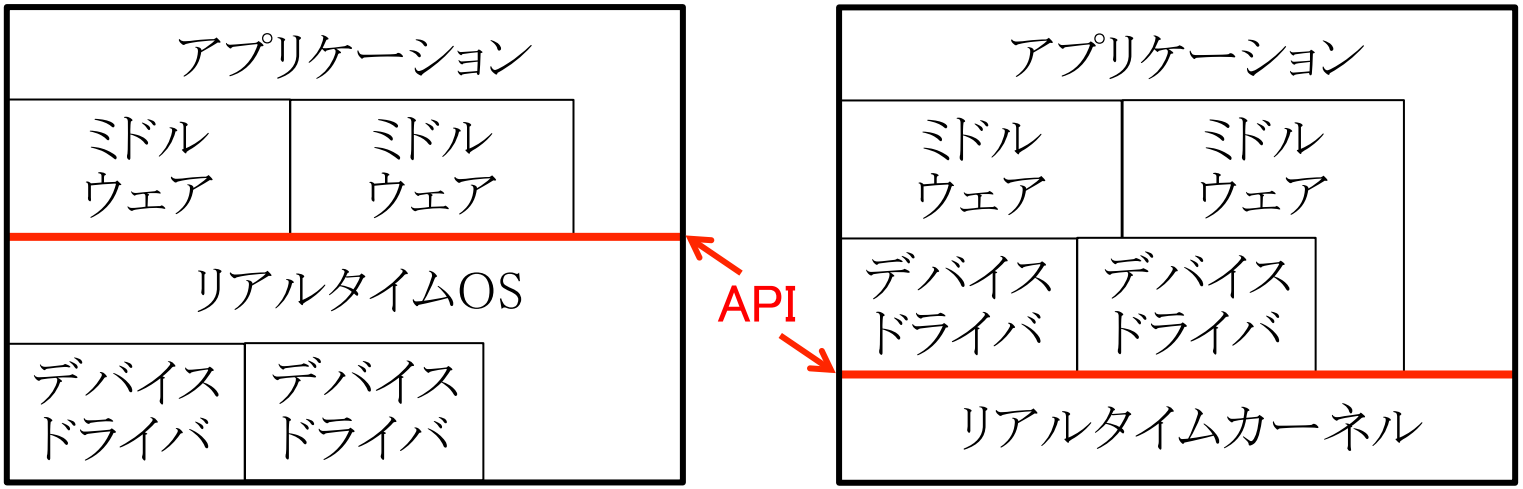
RTOSとリアルタイムカーネル

リアルタイムカーネル

- (元々は)RTOSの中心になるモジュールの意味
- どのようなシステムにも共通する資源を扱う
  - プロセッサ, メモリ, タイマ, …
  - 操作に時間のかかる資源を扱うモジュールは, カーネルの上で実現した方がスマートという理由も
- 保護機能を持たない場合が多い
- リアルタイムモニタ, リアルタイムエグゼクティブと呼ばれることもある
- リアルタイムカーネル相当の機能しか必要としない場合には「リアルタイムカーネル＝RTOS」
- 汎用のOSとはかなり違うものなので, リアルタイムカーネルと呼んで区別する

アーキテクチャによるRTOSの分類（中間的なものもある）

- 汎用OS型
- リアルタイムカーネル型



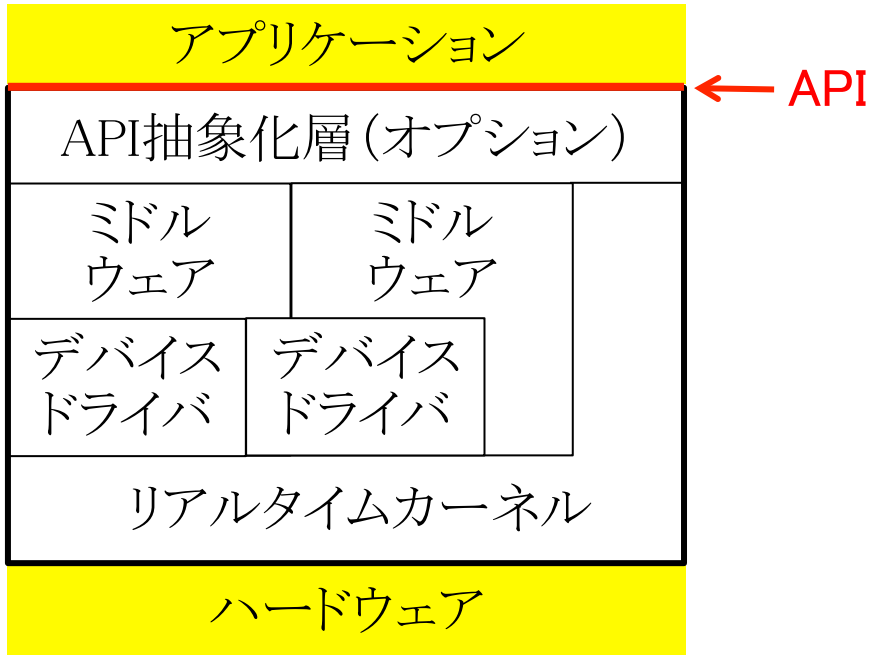
- OSの機能は豊富
- サイズは比較的大きい
- 一般に応答時間は遅い
- デバイスドライバは別のAPIで作成

- カーネルの機能は限定
- カーネルサイズは小さい
- 一般に応答時間は早い
- デバイスドライバとアプリケーションは同じAPI

! このセミナーでは，主にリアルタイムカーネル型を想定

リアルタイムカーネルを核にしたSPF

- リアルタイムカーネルを中心に，あるアプリケーションドメイン向けのデバイスドライバやミドルウェアを載せて，ソフトウェアプラットフォーム（SPF）を構築するケースが多い



- API抽象化層（例えば，POSIXインタフェース層やAUTOSAR仕様のRTE）はあってもなくてもよい

RTOS/SPFの主な機能

リアルタイムカーネルの機能

- マルチタスク機能 → プロセッサの仮想化
- タスク間通信・同期機能
- 時間同期/管理 → タイマの仮想化
- メモリ管理 → メモリの仮想化
- 割込み管理/処理，例外管理/処理，システム管理　など
- 保護機能 … 保護機能を持ったRTOSも増えつつある

汎用OS型のRTOS/SPFのその他の機能

- 入出力管理／デバイス管理機能
- ファイル管理機能（ファイルシステム）
- 通信・ネットワーク機能（プロトコルスタック）
- ユーザインタフェース機能（GUIなど）
- プログラム管理／ローディング機能　などなど

マルチタスク機能

タスクとは？

! ここでのタスクは，スレッドと同義

- プログラムの並行実行の単位
  - 1つのタスク中のプログラムは逐次的に実行される
  - 異なるタスクのプログラムは並行して実行される
- プロセッサを抽象化・多重化したもの

プロセッサ →（抽象化／多重化）→ タスク

… 仮想化されたプロセッサ

マルチタスク機能

- 複数のタスクを疑似並列に実行するための機能
  - シングルプロセッサシステムでは，実際に同時に実行できるタスクは1つのみ
  - 複数のタスクが同時に実行しているかのように見せる

用語の整理

- *ディスパッチ(タスクディスパッチ，タスク切換え)*
  - プロセッサが実行するタスクを切り換えること
  - ディスパッチを実現するOS内のモジュールがディスパッチャ
- *スケジューリング(タスクスケジューリング)*
  - どの時間にどのタスクを実行するかを決定すること
  - 多くのRTOSにおいては，次に実行するタスクを決定する処理
  - スケジューリングを実現するOS内のモジュールがスケジューラ(UNIX/Linuxのスケジューラは，スケジューリングに加えて，ディスパッチも行う)
- *スケジューリングアルゴリズム*
  - どのようにして次に実行するタスクを決定するか？

プリエンプティブな優先度ベーススケジューリング

- ! ほとんどのRTOSで採用されているスケジューリング方式（RTOSによっては他の方式もサポートしている）
- *優先度ベーススケジューリング*
  - 最も優先度の高いタスクが実行される
  - 優先度の高いタスクが実行できなくなるまで，優先度の低いタスクは実行されない
  - 同一優先度タスク間では FCFS（First Come First Served）
- *プリエンプティブスケジューリング*
  - 優先度の高いタスクが実行可能になると，優先度の低いタスクが実行途中でも，タスク切換えが起こる
  - 実行可能になるきっかけは割込み
- ! 汎用OSのスケジューリングとは大きく異なる

マルチタスクの必要性

タスク分割の基本的な考え方

- 独立した処理の流れを独立したタスクに
  - 複数のサブシステムに対する処理
  - 別々の入出力装置/イベントに対する処理　など

リアルタイムシステムにおけるタスク分割の意義

- *論理的な処理の順序と時間的な処理の順序を分離*
  - プログラムは論理的な処理順序で記述
  - RTOSは時間的な処理順序に従って実行する

↓

- プログラムの保守性・再利用性の向上
- 外部で開発されたソフトウェア部品の導入を容易に

論理的な処理順序と時間的な処理順序の分離

例として次の処理を行う場合を考える

- モータの制御処理を10ミリ秒周期で行う．1回の処理に最大5ミリ秒かかる
- それと並行して，ビデオカメラで撮影した画像を認識する処理を行う．1回の処理に最大100ミリ秒かかる

RTOS無しで実現するには…

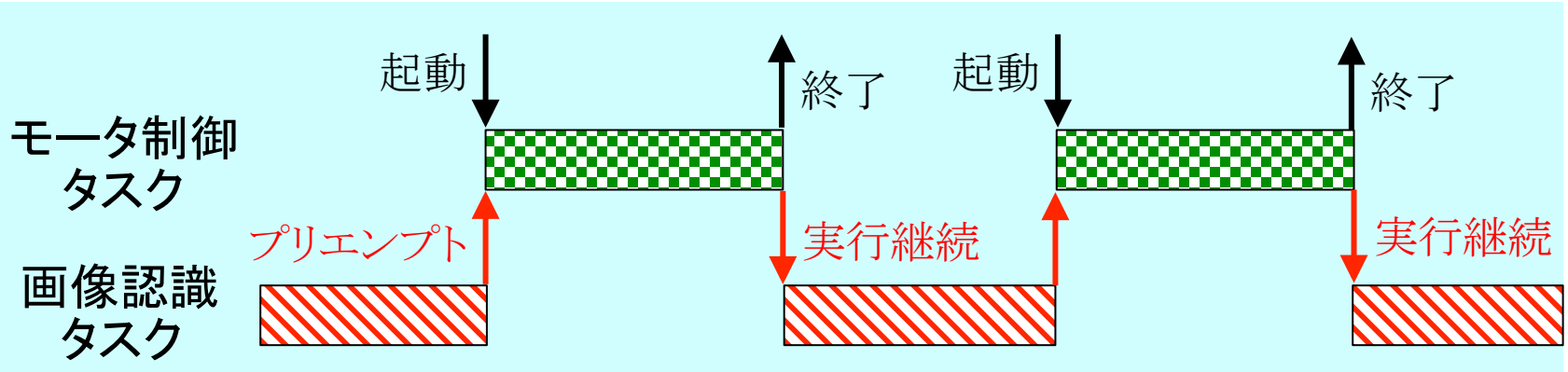
- モータの制御処理を，10ミリ秒周期で回るメインループで実行する
- 画像認識プログラムを，5ミリ秒単位の20個の処理に分割し，メインループの中で1つずつ順に実行する

↓

画像認識プログラムの保守性・再利用性が低下

RTOS（マルチタスク機能）を用いると…

- 2つの処理を別々のタスク（モータ制御タスクと画像認識タスク）で実現
- モータ制御タスクの方に高い優先度を与える
- モータ制御タスクは10ミリ秒周期で起動する



画像認識プログラムを分割する必要はなく，保守性・再利用性が向上

論理的な処理順序と時間的な処理順序の分離が本質

- モータ制御処理と画像認識処理は，論理的には独立の処理
- 時間的には，画像認識処理の途中にモータ制御処理を割り込ませないと間に合わない

↓

- プログラムは論理的な処理順序で（つまり両処理を独立して）記述し，それを時間的な処理順序で実行するのはRTOSに任せる

↓

- *時間制約を持ったプログラムの保守性・再利用性の向上*
- *外部で開発されたソフトウェア部品の導入を容易に*

! ただし，この例の場合には，RTOSを使わずに，メインループと割込み処理で実現する方法もある

タスク状態

基本的なタスク状態

- RTOSでは，タスクが疑似並列実行されている状況をアプリケーション開発者に明示することが必要

実行状態と実行可能状態
(RUNNING)　　(READY)

実行すべき処理が無い状態　休止状態 (DORMANT)

- 低優先度タスクが実行される
- 起動されると，タスクの先頭から実行される

何らかの事象発生を待っている状態　（広義の）待ち状態

! 事象の発生をループで待ってはいけない

- 低優先度タスクが実行される
- 事象が発生した場合に，前の続きから実行される

待ち状態の有用性

- プログラム中のどこを実行しているかで，状態を表現するプログラムで有用
- 特に，呼び出された関数の内部で待ち状態としたい場合

例）

```
サーバからのデータ取出し処理 ( )
{
    サーバにデータAを問い合わせるコマンドを送る;
    サーバからの応答を待ち，変数Aに格納する;
    サーバにデータBを問い合わせるコマンドを送る;
    サーバからの応答を待ち，変数Bに格納する;
}
```

- ローカル変数で状態を保持できるのもメリットの1つ

タスク間の通信と同期

タスク間通信・同期の必要性

- タスクが協調して動作するためには，タスク間でデータをやりとりすること（＝タスク間通信）が必要
- タスク間通信の際には，タスク同士で動作タイミングをあわせること（＝タスク間同期）が必要
- 複数のタスクが同一の資源を取りあう場合にも，タスク間同期が必要（排他制御）

タスク間通信・同期のタイプ

! タスクを仮想化されたプロセッサと捉えるなら，タスク間の通信・同期は，プロセッサ間の通信・同期を仮想化したもの

- 共有メモリによる通信
- メッセージによる通信

共有メモリによる通信

- 複数のタスクからアクセスできるメモリ領域（共有メモリ）上に受け渡しするデータを置く
- 共有データを読み書きする際には，排他制御することが必要．RTOSは排他制御のための機能を持つ
  （共有データが1度に読み/書きできる場合は例外）
  - 割込み/ディスパッチの禁止，タスク優先度の変更
    - ! *マルチプロセッサへの拡張性に注意*
  - セマフォ，ミューテックス
    - ! *デッドロックと優先度逆転に注意*
- 共有データを速やかに処理させたい場合には，事象の発生を知らせる機能を用いる
  - タスクの起動/起床
  - イベントフラグ，条件変数（Condition Variable）

メッセージによる通信

- RTOSが持つメッセージ通信のための機能を用いて，タスク間でメッセージを受け渡しする

メッセージ通信機能のタイプ

- コネクションあり *or* なし，単方向 *or* 双方向，1対1（送受信できるタスクが固定）*or* n対1 *or* n対n
- 通信相手の指定方法
  - 通信オブジェクトを指定 *or* 相手タスクを指定 *or* …
- 同期メッセージ通信 *or* 非同期メッセージ通信
- （非同期通信で）バッファにメッセージが無い時の振舞
  - 待つ（ブロッキング）*or* 待たない（ノンブロッキング）
- （非同期通信で）バッファがフルの時の振舞い
  - 待つ（ブロッキング）*or* 待たない（ノンブロッキング）*or* 上書きする

メッセージ通信機能のタイプ（続き）

- メッセージが固定長 *or* 可変長（任意長）
- （可変長の時に）パケットの単位がある *or* ない
- ポインタを渡す *or* コピーする
  （メッセージが1度に読み/書きできる場合は意味が無い）
- （非同期通信で）メッセージのキューイング順序
  - FIFO順 *or* 優先度順
- 受信/送信待ちタスクのキューイング順序
  - FIFO順 *or* 優先度順
- その他の特殊な機能
  - マルチキャスト/ブロードキャスト
  - 状態メッセージ（OSEK/VDX仕様のunqueued message）

! RTOSの持つメッセージ通信機能（複数の機能を持つ場合も多い）の性質を良く理解することが必要

静的OS (Static Operating System)

! 専用システムであるという特性を活かしたOS技術

静的OSとは？

- 使用するOS資源（タスク，セマフォなど）を静的に（設計時に）定義するOS

  例）多くのITRON仕様準拠カーネル

  OSEK/VDX仕様OS，AUTOSAR OS仕様（OS資源を動的に生成するAPIが仕様に規定されていない）

静的OSの利点

- メモリ容量（特にRAM容量）を小さくできる
- システム起動時間を短縮できる
- システムの動作中にメモリ不足になることがない
- 静的な情報を使った最適化が可能

コンフィギュレーション記述の方法

(1) コンフィギュレーション記述の言語を定め，そこからツールにより構成・初期化ファイルを生成する方法
- ITRON仕様の静的API
- OSEK/VDX仕様のOIL（OSEK Implementation Language）
- AUTOSAR仕様ではXMLのフォーマットを規定

(2) GUIベースのコンフィギュレーションツールを用意する方法

(3) 構成・初期化ファイルを直接記述する方法（C言語の初期化文の形で記述するのが一般的）
- 静的な情報を使った最適化は難しい

RTOSを使用するメリット

- ソフトウェアの構造化による生産性，保守性，信頼性の向上

  ソフトウェアの構造化 ≒ モジュール化

- 時間制約を持った多重処理システムの構築を容易に
  - 論理的な処理順序と時間的な処理順序の分離により，時間制約を持ったソフトウェアの保守性・再利用性が向上

    ➡ ソフトウェア部品を用いたソフトウェア開発（コンポーネントベース開発）の基盤

- ハードウェア（特にプロセッサ）の違いを隠蔽
  - ハードウェアの細部を知らなくても，アプリケー ションが構築できる
  - 「リアルタイムカーネルは，プロセッサのデバイスドライバである」

RTOSを使用するデメリット

RTOSを使用するメリットとの引き換え

- RTOS自身のオーバヘッド
  - オーバヘッドによる実行性能の低下
  - RTOS自身のワークエリアによるメモリの消費

  → リアルタイムカーネルでは，半導体技術やRTOS実装技術の進歩により，問題になる場面は減ってきた
- プリエンプティブスケジューリングのデメリット
  - タスクのスタックエリアによるメモリの消費
  - 非決定性が増すことによるデバッグ・テストの困難化

  → 行儀のよいタスク間同期・通信の設計が必要
- RTOSのブラックボックス化による解析の困難性

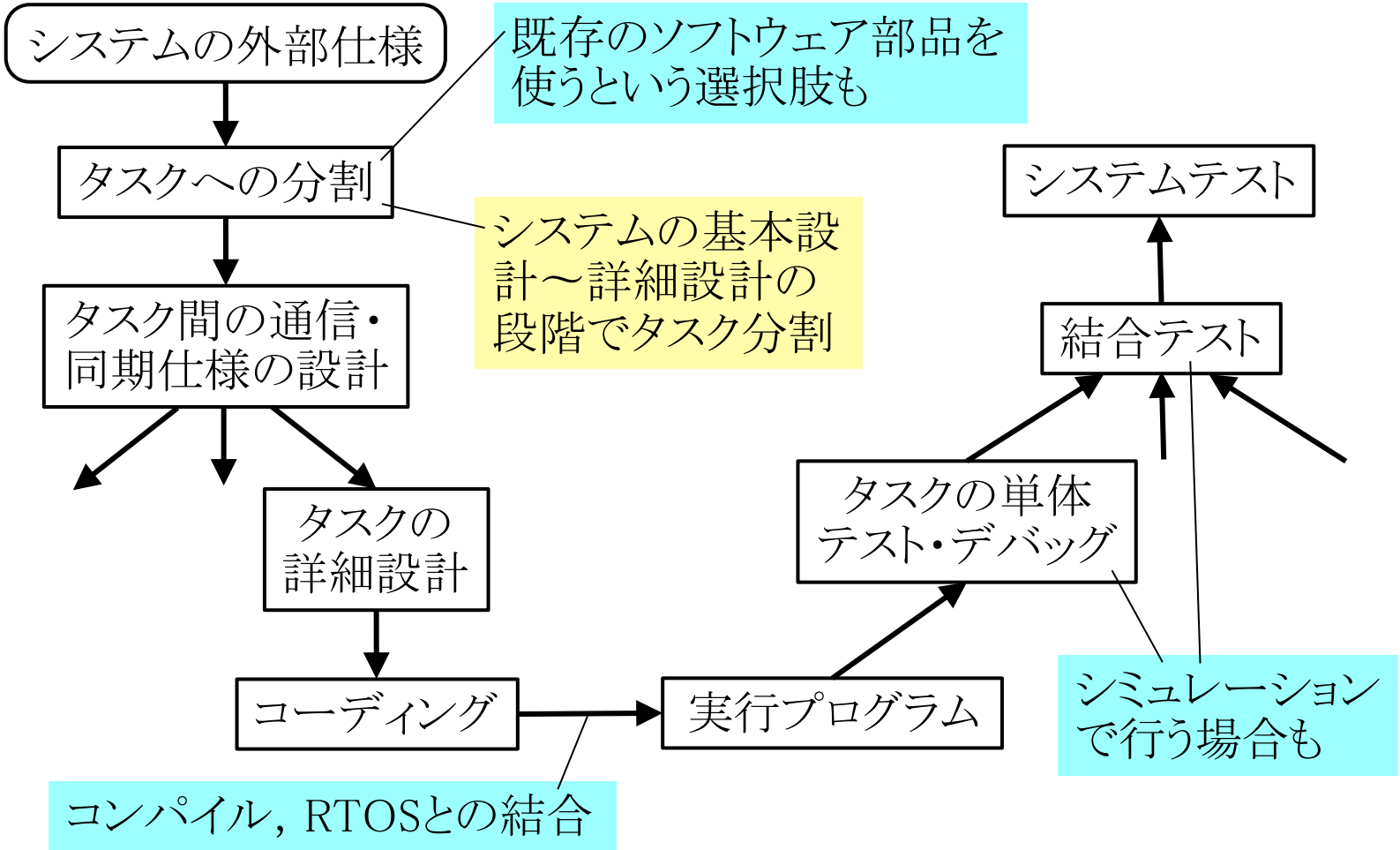
  → RTOSに対応したツールの利用でかなり改善できる

  ! RTOSの原理・内部構造を知るべき

RTOSを用いたシステム設計の指針

リアルタイムOSを用いたソフトウェア開発の流れ

典型的な開発フロー（要求分析工程は除く）



タスクへの分割指針

タスクへの分割

- どのようなアプリケーションにも適用できる一般的な手法は存在しない
- 次のような要因を考慮に入れて決定すべき
  - アプリケーションの機能と性能要件
  - ソフトウェア部品として外部から導入する部分
  - ソフトウェアの開発体制

リアルタイム性向上のための分割指針

- ! 論理的な処理順序と時間的な処理順序の分離により，時間制約を持ったプログラムの保守性が向上
- デッドラインの異なる処理は別タスクに
- （過負荷になる場合）重要度の異なる処理は別タスクに

モジュール化のための分割指針

! ソフトウェアの構造化・開発容易化のための分割

- 異なる機能モジュールや異なるデバイスの操作は別タスクに
- 異なる種類の処理は別タスクに
  - I/Oタスクと内部処理タスクに分割
  - システムの監視・保守・診断の処理を別タスクに
  - 例外時・故障時の処理を別タスクにする方法もある（別タスクにしない方法もある）
- 再利用しやすい単位は単独のタスクに
- まとまった処理単位は同一タスク内に
  - 一つの状態遷移／一つの結果を出す処理／データの流れが綴じた処理は同一タスク内で
  - 状態（モード）毎に別タスクにする方法もある

モジュール化のための分割指針 – 続き

- 同一処理を複数並行して行いたい場合
  - 同一のプログラムを共有して，複数のタスクを動作させる方法
  - データで切り分ける方法もある
- 開発者／グループ毎にタスクを分割する
- 起動イベント／起動タイミングの異なる処理
  - 起動イベント／タイミング毎にタスクを分ける方法
  - 起動周期毎にタスクを分ける方法は一般的
  - 起動イベント／タイミングが異なっていても，まとまった処理単位は同一タスクとする方法も

設計上の留意点

- 分割しすぎはオーバヘッドの増大につながるので注意

タスクの優先度の決定

優先度の決め方

- 時間制約を満たせる範囲では,

  急ぐタスク(デッドラインの短いタスク)を優先

- 一時的な過負荷時に時間制約を満たせない場合には,

  重要なタスク(時間制約を満たせなかった場合の被害が大きいタスク)を優先

時間制約を満たせるかをどう判断するか？

- リアルタイムスケジューリング理論を適用する

一時的な過負荷時にはどうするか？

- 重要性の低いタスクをさぼる／精度を落とす

  QoS (Quality of Service) 制御

- それがダメなら高性能なハードウェアを用いるしかない

タスク間通信・同期の設計

共有メモリ vs. メッセージ通信

- 基本的には，一方で実現できることは，もう片方でも実現できる
  - 一般的には，共有メモリの方がオーバヘッドが小さい
  - システム検証には，通信の粒度が大きく，RTOSレベルで監視できるメッセージ通信の方が扱いやすい

    → 例えば，問題の切り分けが容易
- 状況により，どちらかが有利／便利な状況がある
  - 情報の流れが 1:n の場合（ブロードキャストを含む）や，情報が必要なタイミングが不定の場合には，共有メモリが有利
  - 情報のキューイングが必要な場合には，メッセージ通信機能が便利

設計上の留意事項

- 両方式を混在させて使うと，システム設計が複雑になって，見通しが悪くなる可能性がある
- セマフォ／ミューテックスによる排他制御では，デッドロックと優先度逆転に関して注意が必要
- メッセージ通信によりサーバタスクに処理依頼する場合にも，優先度逆転の問題が起こりうる

デッドロック（deadline）

- 2つのタスクが，互いに相手の処理が進むのを待つ状態．3つ以上のタスク間でも発生
  - 典型的には，2つ以上のセマフォ／ミューテックスを，タスクによって異なる順序でロックする場合に発生
  - ロックする順序を決めれば解決（常にできるとは鍵らない）

テスト (検証) とデバッグ

タスク／割込みハンドラの単体テスト

- 当該タスク／割込みハンドラを含んだ最小構成でリンク
- 他のタスクの振舞いはシミュレーションで
- タスク／割込ハンドラの処理時間，スタック使用量などを評価する

  など

複数タスクの結合テスト

- シミュレーションされているタスクを，単体テスト」済みのタスクに入れ換えていく
- タスク間のインタフェースの確認
- 資源の競合テスト（排他制御が正しいか？）
- *などなど*

TOPPERS/HRP2カーネルのAPI

TOPPERS新世代カーネルとは？

TOPPERS新世代カーネルの位置付け

- ITRON仕様を拡張・改良して，TOPPERSプロジェクトで開発された一連のRTOS
  - ITRON仕様のサポート範囲内では，ITRON仕様との差は小さい（ITRON仕様のバージョン間の差と同程度）
  - ITRON仕様に含まれない，マルチコア向け拡張を含む
- 以下のリアルタイムカーネルが，TOPPERS新世代カーネルに含まれる
  - TOPPERS/ASPカーネル … 標準セット
  - TOPPERS/SSPカーネル … 最小セット
  - TOPPERS/FMPカーネル … マルチコア向け拡張
  - TOPPERS/HRP2カーネル … 保護機能拡張

TOPPERS新世代カーネル仕様の設計方針

(1) μITRON4.0仕様をベースに拡張・改良を加える

- 多くの実績があるμITRON4.0仕様をベースに
- μITRON4.0仕様の不十分な点は積極的に拡張・改良

(2) ソフトウェアの再利用性を重視する

- ソフトウェアの再利用性向上のために，少々のオーバヘッドがあっても，ターゲット依存項目を減らす

(3) 高信頼・安全なシステム構築を支援する

- アプリケーションに誤用されにくい仕様とする
- 妥当なオーバヘッドで救済できる誤用は救済する

(4) アプリケーション構築に必要な機能は積極的に取り込む

- 多くのアプリケーションに共通に必要な機能を実装
- ただし，(1)～(3)の方針を満たすことが前提

TOPPERS新世代カーネル統合仕様書

- TOPPERS新世代カーネルに属する一連のリアルタイムカーネルの仕様を，統合的に記述した仕様書
- 以下のURLからダウンロード可能
  - http://www.toppers.jp/documents.html

TOPPERS新世代カーネル統合仕様書の構成

第1章 TOPPERS新世代カーネルの概要

第2章 主要な概念と共通定義

- 複数の機能単位にまたがる概念や共通の定義

第3章 システムインタフェースレイヤAPI仕様

- システムコールインタフェースレイヤ（SIL）のAPI仕様

第4章 カーネルAPI仕様

- カーネルのサービスコールと静的APIのAPI仕様

第5章 リファレンス

第2章 主要な概念と共通定義

- 2.1 仕様の位置付け
- 2.2 APIの構成要素とコンベンション
- 2.3 主な概念
- 2.4 処理単位の種類と実行
- 2.5 システム状態とコンテキスト
- 2.6 タスクの状態遷移とスケジューリング規則
- 2.7 割込み処理モデル
- 2.8 CPU例外処理モデル
- 2.9 システムの初期化と終了
- 2.10 オブジェクトの登録とその解除
- 2.11 オブジェクトのアクセス保護
- 2.12 システムコンフィギュレーション手順
- 2.13 TOPPERSネーミングコンベンション
- 2.14 TOPPERS共通定義
- 2.15 カーネル共通定義

第4章 カーネルAPI仕様

- 4.1 タスク管理機能
- 4.2 タスク付属同期機能
- 4.3 タスク例外処理機能
- 4.4 同期・通信機能
- 4.5 メモリプール管理機能
- 4.6 時間管理機能
- 4.7 システム状態管理機能
- 4.8 メモリオブジェクト管理機能
- 4.9 割込み管理機能
- 4.10 CPU例外管理機能
- 4.11 拡張サービスコール管理機能
- 4.12 システム構成管理機能

TOPPERS/HRP2カーネルとは？

適用対象（ASPカーネルの適用対象と比べて）

- さらに高い信頼性・安全性を要求される組込みシステム
- より大規模な（プログラムサイズ（バイナリコード）が数百KB以上のシステム）組込みシステム

ハードウェア要件

- 特権モードと非特権モードを持つこと
- メモリ管理ユニット（MMU）またはメモリ保護ユニット（MPU）を持つこと

2つの拡張パッケージを持つ

- 動的生成機能拡張パッケージ … EV3RTで使用
- メッセージバッファ機能拡張パッケージ … EV3RTには未適用

HRP2カーネルの仕様

TOPPERS新世代カーネル統合仕様書とHRP2カーネル

- HRP2カーネルの仕様は，TOPPERS新世代カーネル統合仕様書に記述されている
- ただし，HRP2カーネルは，統合仕様書に記述されている機能をすべてサポートしているわけではない

HRP2カーネルがサポートする機能セット

- HRP2カーネルは，「保護機能対応カーネル」であり，「マルチプロセッサ対応カーネル」，「動的生成対応カーネル」ではない
  - ただし，動的生成機能拡張パッケージを用いると（EV3RTは用いている），動的生成対応カーネルの機能の一部をサポートする

統合仕様書中の適用される記述/適用されない記述

- 「保護機能対応カーネルでは」で始まる記述は（基本的には）適用される
- 「マルチプロセッサ対応カーネルでは」「動的生成対応カーネルでは」で始まる記述は適用さない
  - ただし，動的生成機能拡張パッケージを用いると（EV3RTは用いている），「動的生成対応カーネルでは」で始まる記述の一部は適用される
- 【TOPPERS/HRP2カーネルにおける規定】の項に書かれていることは，（文字通り）適用される
  - HRP2カーネルにおける例外規定は，この項の中に書かれている
  - 動的生成機能拡張パッケージを用いた場合のことも，この項の中に書かれている

APIのサポート種別の記号（抜粋）

- 〔P〕は保護機能対応カーネルのみでサポートされるAPI
- 〔p〕は保護機能対応でないカーネルのみでサポートされるAPI
- 〔M〕〔D〕は，それぞれ，マルチプロセッサ対応カーネル，動的生成対応カーネルのみでサポートされるAPI

APIのサポート種別の記号の実際の例

- act_tsk〔T〕… HRP2カーネルでサポートされる
- dis_wai〔TP〕… HRP2カーネルでサポートされる
- snd_mbx〔Tp〕… HRP2カーネルでサポートされない
- mact_tsk〔TM〕… HRP2カーネルでサポートされない
- acre_tsk〔TD〕… HRP2カーネルでサポートされない．ただし，動的生成機能拡張パッケージを用いると，サポートされる場合も（個別に記述）

基本的な用語

オブジェクト(カーネルオブジェクト)

- カーネルが管理対象とするソフトウェア資源
- 種類毎に，番号によって識別する

処理単位

- 対応するプログラムを持つオブジェクト(または，そのオブジェクトに対応付けられたプログラム)
- プログラムは，アプリケーションで用意し，カーネルが実行制御する

タスク

- カーネルが実行制御するプログラムの並行実行の単位
- 処理単位の一種

自タスク

- サービスコールを呼び出したタスク

ディスパッチ（タスクディスパッチ）

- プロセッサが実行するタスクを切り換えること

ディスパッチの保留

- ディスパッチが起こるべき状態となっても，何らかの理由でディスパッチを行わないこと
- その理由が解除された時点で，ディスパッチが起こる

スケジューリング（タスクスケジューリング）

- 次に実行すべきタスクを決定する処理

優先順位

- 処理単位の実行順序を説明するための仕様上の概念
- 処理単位は，優先順位の高い順に実行される

優先度

- 処理単位の優先順位やメッセージの配送順序などを決定するために，アプリケーションが与えるパラメータ

割込み（外部割込み）

- プロセッサが実行中の命令とは独立に発生するイベントによって起動される例外処理

割込みのマスク（禁止）

- 周辺デバイスからの割込み要求をプロセッサに伝える経路を遮断し，割込み要求が受け付けられるのを抑止すること
- マスクが解除された時点で,まだ割込み要求が保持されていれば，その時点で割込み要求を受け付ける

NMI（Non-Maskable Interrupt）

- マスクすることができない割込み

CPU例外

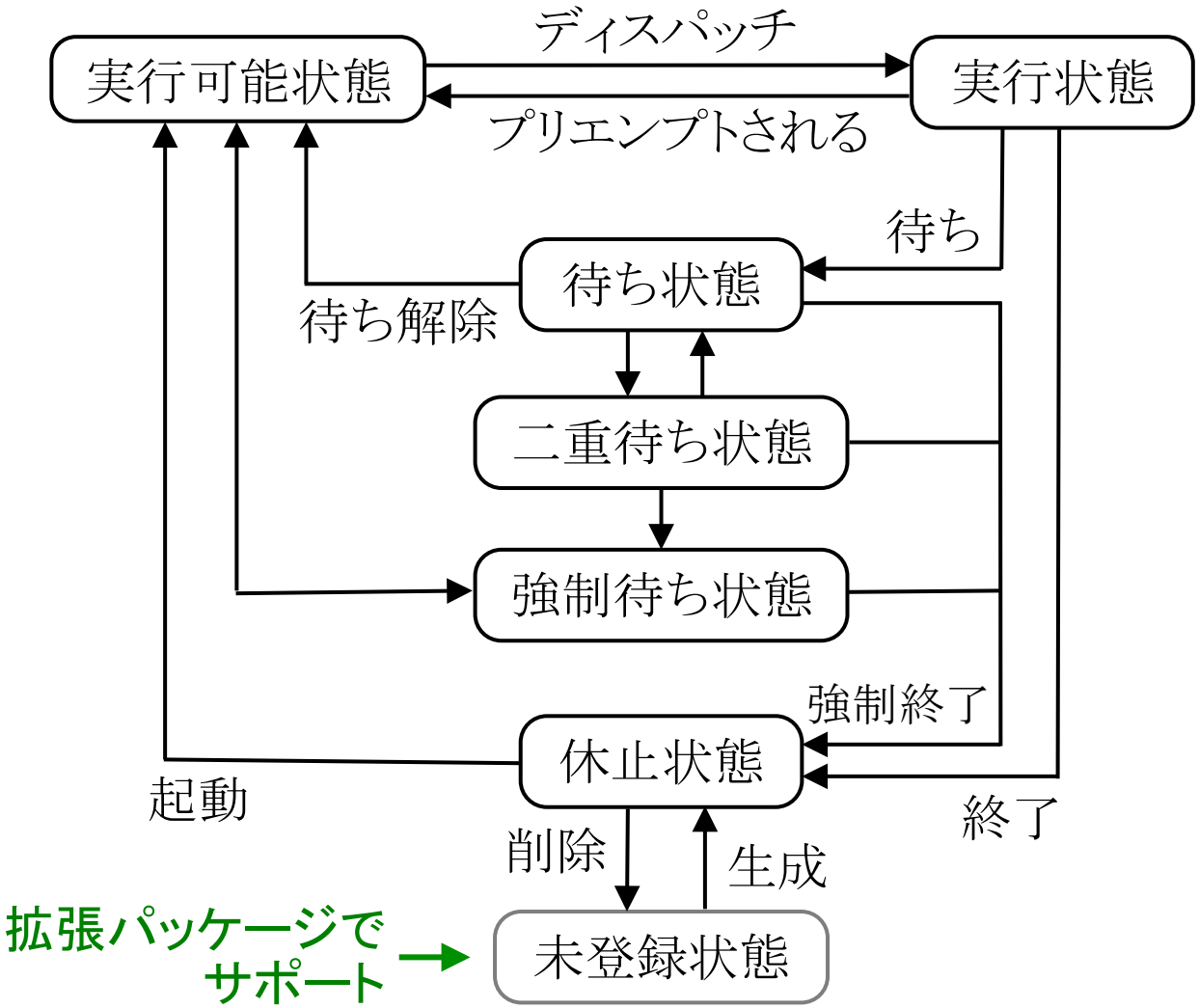
- プロセッサが実行中の命令に依存して起動される例外処理

タスクの状態遷移とスケジューリング規則

タスク状態

- 実行できる状態（runnable）
  - 実行状態（running）
  - 実行可能状態（ready）
- 休止状態（dormant）
- 広義の待ち状態（blocked）
  - （狭義の）待ち状態（waiting）
    … タスクが自ら実行を止めている状態
  - 強制待ち状態（suspended）
    … タスクが他のタスクによって実行を止められた状態
  - 二重待ち状態（waiting-suspended）
- 未登録状態（non-existent） ← 拡張パッケージでサポート

タスクの状態遷移(一部省略)



- ディスパッチ
- プリエンプト
- 起動
- 終了
- 強制終了
- 待ち
- 待ち解除
- 強制待ち
- 強制待ちからの再開
- 生成
- 削除

タスクのスケジューリング規則

- タスクに与えられた優先度に基づくプリエンプティブな優先度ベーススケジューリング
- 同優先度のタスクは，先に実行できる状態（実行状態または実行可能状態）になったタスクを先に実行するFCFS（First Come First Served）方式でスケジューリング
  - プリエンプトされても，実行順序は後回しにならない
  - 待ち状態になったタスクの実行順序は最後になる
- 優先度割付けは静的が基本だが，優先度を動的に変更する機能も用意
- 同優先度のタスク内での優先順位を変更する機能を用意
  - これを用いて，同優先度内でのラウンドロビンスケジューリングも実現可能
- HRP2カーネルでは，優先度は16段階

処理単位と優先順位

処理単位の種類

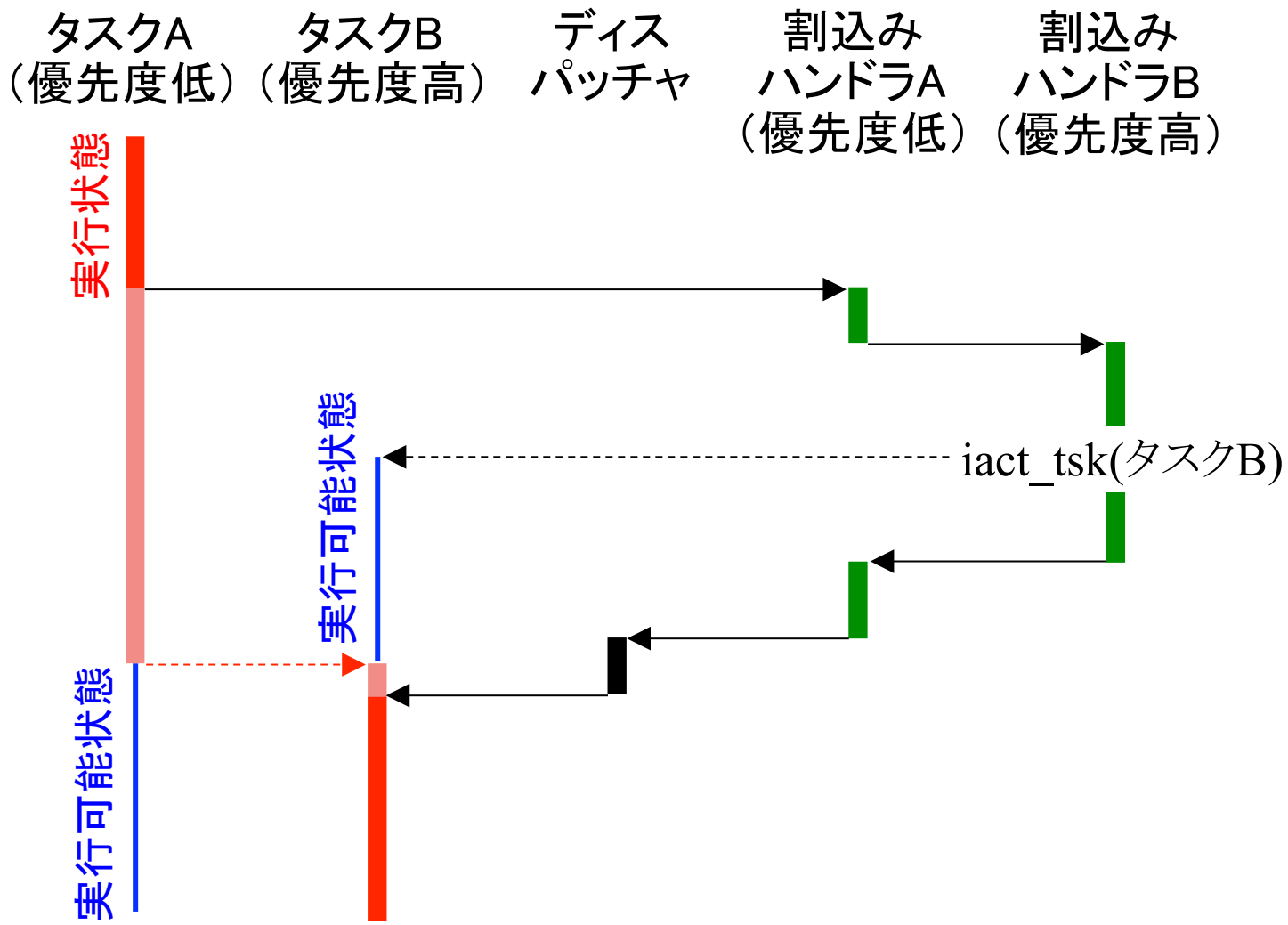
- タスク
  - タスク例外処理ルーチン
- 割込みハンドラ
  - 割込みサービスルーチン（ISR）
  - タイムイベントハンドラ
    - 周期ハンドラ
    - アラームハンドラ
    - オーバランハンドラ
- CPU例外ハンドラ
- 拡張サービスコールルーチン
- 初期化ルーチン
- 終了処理ルーチン

処理単位の優先順位

- 次の順序で優先順位が高い
  - 割込みハンドラ(ISR, タイムイベントハンドラを含む)
  - ディスパッチャ
  - タスク
- 割込みハンドラの優先順位はディスパッチャよりも高いことから，割込みハンドラが動いている間は，タスク切換えは起こらない

  → 遅延ディスパッチの原則
- CPU例外ハンドラの優先順位
  - CPU例外がタスクで発生した場合には，ディスパッチャと同じ優先順位で，ディスパッチャより先に実行
  - その他の処理単位で発生した場合には，その処理単位と同じ優先順位で，その処理単位より先に実行

タスク切換えの遅延（遅延ディスパッチの原則）



システム状態とコンテキスト

タスクコンテキスト

- タスクの処理の一部と見なすことのできるコンテキストの総称
  - タスク(タスク例外処理ルーチンを含む)
  - タスクコンテキストから呼び出した拡張サービスコールルーチン

非タスクコンテキスト

- タスクコンテキストに含まれないコンテキストの総称
  - 割込みハンドラ(ISR, タイムイベントハンドラを含む)
  - CPU例外ハンドラ
  - 非タスクコンテキストから呼び出した拡張サービスコールルーチン

CPUロック状態

- すべての割込み(カーネルの管理外のものを除く)が禁止され，ディスパッチも起こらない状態
- 呼び出せるサービスコールに制限

ディスパッチ禁止状態

- ディスパッチが起こらない状態
- 自タスクを広義の待ち状態にする可能性のあるサービスコールは呼び出せない

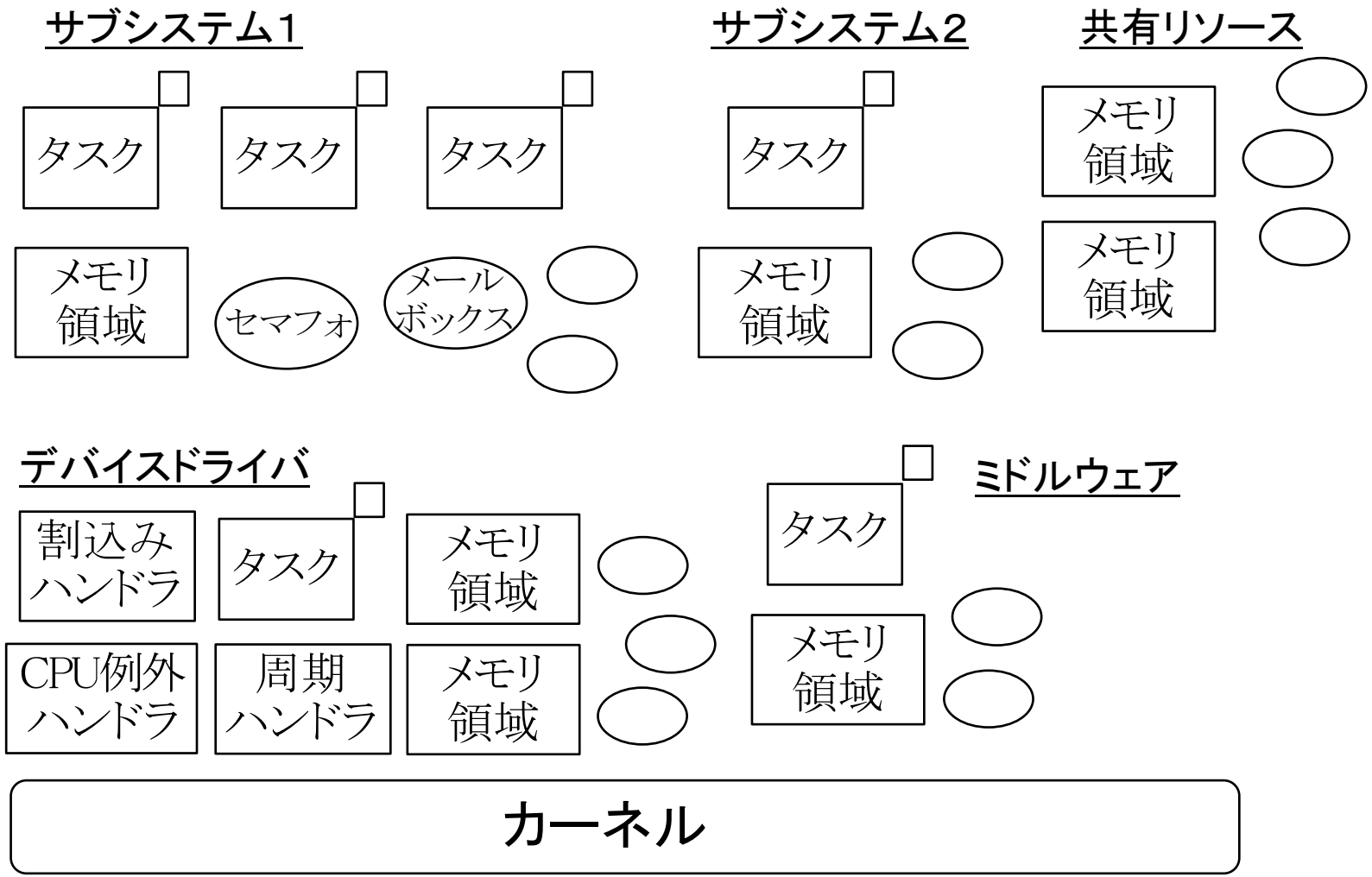
割込み優先度マスク

- 割込み優先度を基準に割込みをマスクするための機構
- 全解除でない時は，ディスパッチは起こらない

ディスパッチ保留状態

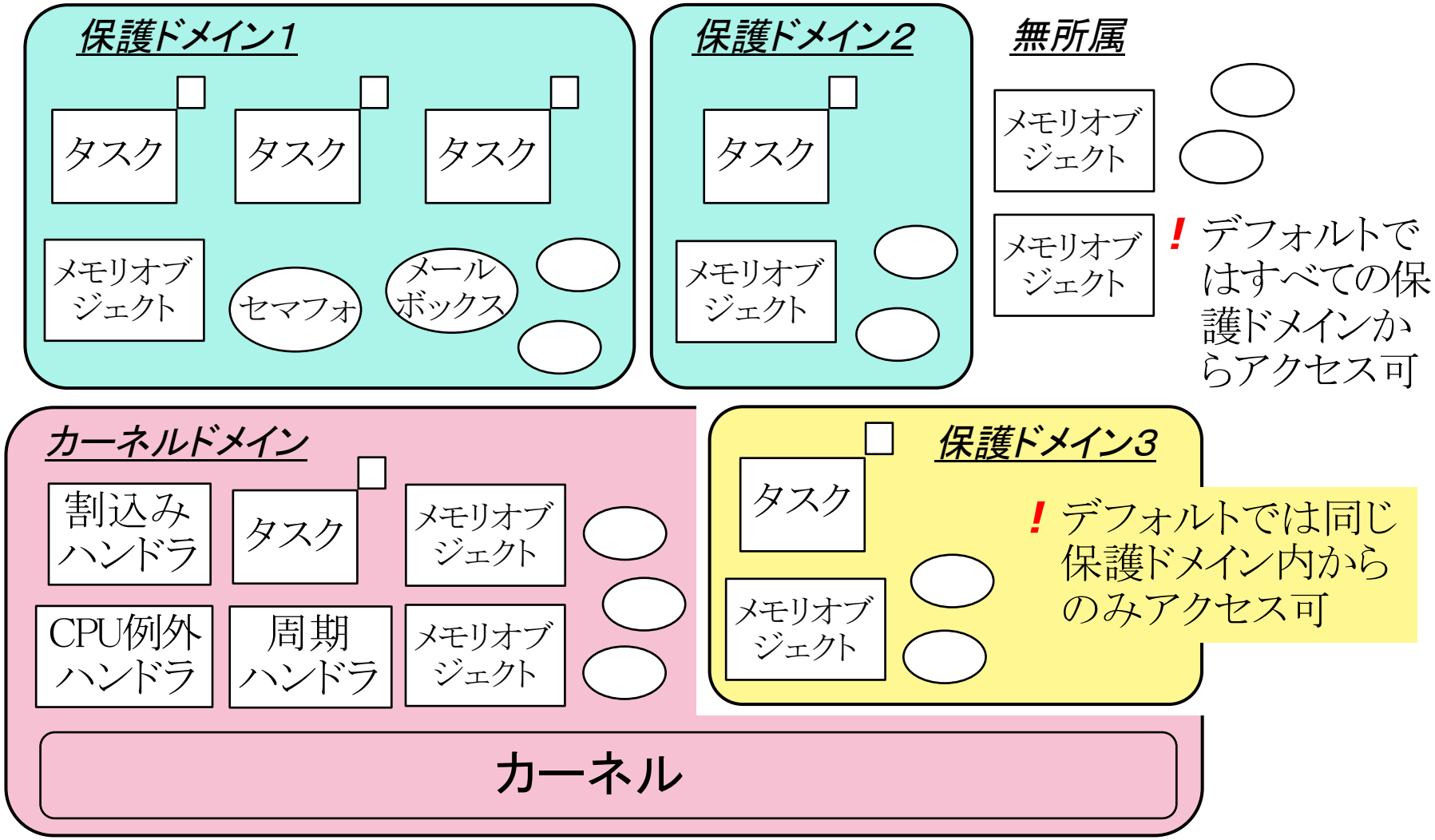
- 何らかの理由でディスパッチが起こらない状態(ディスパッチが起こらない状態の総称)

保護機能の導入イメージ (導入前)



保護機能の導入イメージ (導入後)

! 囲みに穴をあける機能も持つ



保護機能に関する主な概念

アクセス保護

- 処理単位(タスクと考えてよい)が，許可されたカーネルオブジェクトに対して，許可された種別のアクセスを行うことのみを許し，それ以外のアクセスを防ぐ機能
- アクセス制御の用語では，
  - 処理単位 ＝ 主体(subject)
  - カーネルオブジェクト ＝ 対象(object)

メモリオブジェクト(memory object)

- アクセス保護の対象とする連続したメモリ領域で，カーネルオブジェクトの一種
- 互いに重なりあうことはない
- 先頭番地によって識別(先頭番地がオブジェクト番号)
- 先頭番地とサイズにターゲット定義の制約が課せられる

保護ドメイン（protection domain）

- 保護機能を提供するためのカーネルオブジェクトの集合
- 保護ドメインIDによって識別する
- 処理単位は，いずれか1つの保護ドメインに属する
- 他のカーネルオブジェクトは，いずれか1つの保護ドメインに属するか，いずれの保護ドメインにも属さない（無所属のカーネルオブジェクト）

保護ドメインによるアクセス保護

- 処理単位がカーネルオブジェクトにアクセスできるかどうかは，処理単位が属する保護ドメインにより決まるのが原則
- ただし，タスクのユーザスタック領域は，そのタスクのみがアクセスできる
- デフォルトでは，処理単位は，同じ保護ドメインに属するカーネルオブジェクトと，無所属のカーネルオブジェクトのみにアクセスできる

カーネルドメイン（kernel domain）

- システムに1つ存在
- カーネルドメインに属する処理単位は，
  - プロセッサの特権モードで実行される
  - すべてのカーネルオブジェクトに対して，すべての種別のアクセスを行える

ユーザドメイン（user domain）

- ユーザドメインに属する処理単位は，
  - プロセッサの非特権モードで実行される
  - どのカーネルオブジェクトに対してどの種別のアクセスを行えるかを制限できる
- 登録できるユーザドメインの最大数は32

システムタスク（system task）

- カーネルドメインに属するタスク

ユーザタスク（user task）

- ユーザドメインに属するタスク

アクセス許可パターン（access permission pattern）

- アクセスが許可されている保護ドメインの集合を表現するビットパターン（各ビットが1つのユーザドメインに対応）
- ACPTN型（符号無し32ビット整数）で保持
- 以下のマクロと定数を用意
  - TACP(domid) … domid（とカーネルドメイン）のみアクセス可能
  - TACP_KERNEL … カーネルドメインのみアクセス可能
  - TACP_SHARED … すべての保護ドメインからアクセス可能

アクセス許可ベクタ(access permission vector)

- あるカーネルオブジェクトに対する4つの種別のアクセスに関するアクセス許可パターンをひとまとめにしたもの
  - カーネルオブジェクトに対するアクセスは，カーネルオブジェクトの種類毎に，通常操作1，通常操作2，管理操作，参照操作の4つの種別に分類(次のスライド参照)
- 次のように定義されるACVCT型で保持

```
typedef struct acvct {
        ACPTN   acptn1;    /* 通常操作1のアクセス許可パターン */
        ACPTN   acptn2;    /* 通常操作2のアクセス許可パターン */
        ACPTN   acptn3;    /* 管理操作のアクセス許可パターン */
        ACPTN   acptn4;    /* 参照操作のアクセス許可パターン */
} ACVCT;
```

カーネルオブジェクトに対するアクセスの種別（抜粋）

	通常操作1	通常操作2	管理操作	参照操作
メモリオブジェクト	書込み	読出し 実行	det_mem sac_mem	ref_mem prb_mem
タスク	act_tsk can_act wup_tsk can_wup …	ter_tsk chg_pri rel_wai sus_tsk ras_tex …	del_tsk sac_tsk def_tex	get_pri ref_tsk ref_tex ref_ovr
セマフォ	sig_sem	wai_sem pol_sel twai_sem	del_sem ini_sem sac_sem	ref_sem
周期ハンドラ	sta_cyc …	stp_cyc	del_cyc sac_cyc	ref_cyc
システム状態	rot_rdq …	loc_cpu …	def_inh …	get_tim …

サービスコール

- 上位階層のソフトウェア（例：アプリケーション）から，下位階層のソフトウェア（例：カーネル，システムサービス）を呼び出すインタフェース

サービスコールの名称

- カーネルのサービスコール名は，*xxx_yyy*または*zxxx_yyy*の形とする

  例）act_tsk

  操作対象を表す．この場合はタスク（task）

  操作方法を表す．この場合は起動（activate）

  例）slp_tskとtslp_tsk

- 非タスクコンテキストから呼び出すためのサービスコールは，先頭に“i”をつけて区別する

  例）iact_tsk

サービスコールの例：act_tsk（抜粋）

【C言語API】

ER ercd = act_tsk (ID tskid) ;

ER ercd = iact_tsk (ID tskid) ;

【パラメータ】

ID　　tskid　対象タスクのID番号

【リターンパラメータ】

ER　　ercd　正常終了（E_OK）またはエラーコード

【エラーコード】

E_CTX　　コンテキストエラー（……）

E_ID　　不正ID番号（tskidが不正）

E_QOVR　　キューイングオーバフロー（……）

【機能】

tskidで指定したタスク（対象タスク）に対して起動要求を行う．
具体的な振舞いは以下の通り．（以下略）

静的API

- オブジェクトの生成情報や初期状態などを定義するために，システムコンフィギュレーションファイル中に記述するためのインタフェース

静的APIの文法とパラメータ

- C言語の関数呼出しと同様の文法で記述．ただし構造体（へのポインタ）は，各フィールドの値を { } で囲んで列挙
- サービスコールと同等の機能を持つ静的APIは，サービスコール名を大文字にした名称とし，パラメータも同一とする
  - → サービスコールと静的APIを個別に覚える必要がなくなる
- 静的APIのパラメータは何が記述できるかで4種類に分類
  - 一部のパラメータを除いて，式を記述することができる

静的APIの例

```
CRE_TSK( ID tskid, { ATR tskatr, intptr_t exinf,
        TASK task, PRI itskpri, SIZE stksz, STK_T stk } );
```

対応するサービスコール

```
ER ercd = cre_tsk ( ID tskid, T_CTSK *pk_ctsk ) ;
```

静的APIの記述例

```
CRE_TSK( TASK1, { TA_ACT, 0,
        task_main, 10, STACK_SIZE, NULL } );
```

別のヘッダファイルで
値を定義しておく

コンフィギュレータが
スタック領域を割り付ける

コンフィギュレータがタスクIDを割り付ける

オブジェクトが所属する保護ドメインの指定

- オブジェクトを登録する静的APIを，オブジェクトを属させる保護ドメインの囲みの中に記述
- 無所属のオブジェクトを登録する静的APIは，保護ドメインの囲みの外に記述

例）

TASK_AとSEM_Aは DOM_Aに属する

SEM_Aのアクセス許可ベクタを設定

```
DOMAIN (DOM_A) {
        CRE_TSK(TASK_A, { TA_NULL, 1, taskA_main, … });
        CRE_SEM(SEM_A, { TA_NULL, 1, 1 });
        SAC_SEM(SEM_A, { TA_DOM(DOM_A), TA_SHARED,
                                TA_KERNEL, TA_SHARED });
}
KERNEL_DOMAIN {
        CRE_TSK(TASK_K, { TA_ACT, 0, taskK_main, … });
}
CRE_SEM(SEM_S, { TA_TPRI, 1, 1 });
```

TASK_Kはカーネルドメインに属する

SEM_Sは無所属

システムコンフィギュレーションファイルの記述例

```
DOMAIN (DOM_A) {
        CRE_TSK(TASK_A, { TA_[illegible], 1, taskA_main, … });
        CRE_SEM(SEM_A, { TA_NULL, 1, 1 });
        ATT_MOD("obj_a1.o");
        ATA_MOD("obj_a2.o", { TA_DOM(DOM_A), TA_SHARED,
                                TA_KERNEL, TA_SHARED });
}
KERNEL_DOMAIN {
        CRE_TSK(TASK_K, { TA_ACT, 0, taskK_main, … });
        ATT_ISR({ TA_NULL, 1, INTNO_SIO1, sio_isr, 1 });
        ATT_MOD("obj_k.o");
}
CRE_SEM(SEM_S, { TA_TPRI, 1, 1 });
ATT_MOD("shared_lib.a");
```

obj_a1.oは，DOM_Aの専有領域に配置

obj_a2.oは，DOM_Aの専有ライト共有リード領域に配置

obj_k.oは，カーネルドメインの専有領域に配置

shared_lib.aは，共有領域に配置

コンフィギュレーション手順

コンフィギュレーションの流れ

appl.cfg

システムコンフィギュレーションファイル

```
#include “appl.h”
CRE_TSK(TASK1, { TA_ACT, …
CRE_SEM(SEM1, { TA_TPRI, …
```

コンフィギュレータ

内部的には複雑なステップ

kernel_cfg.c

カーネル構成・初期化ファイル

kernel_cfg.h

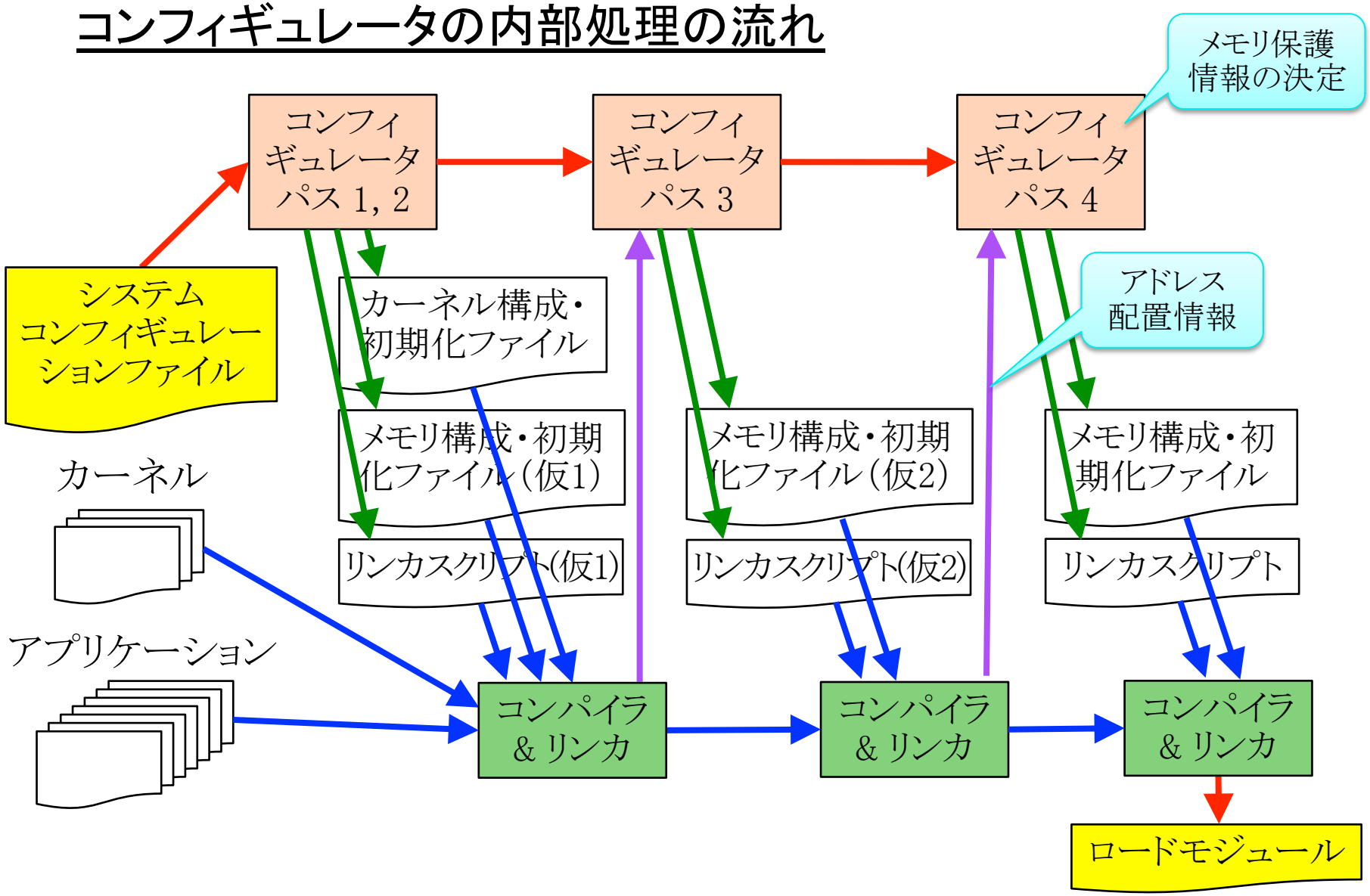
カーネル構成・初期化ヘッダファイル

```
#define  TASK1  2
#define  SEM1   5
```

kernel_mem.c

メモリ構成・初期化ファイル

コンフィギュレータの内部処理の流れ



TOPPERS新世代カーネルの機能一覧

- タスク管理機能
- タスク付属同期機能
- タスク例外処理機能
- 同期・通信機能
  - セマフォ
  - イベントフラグ
  - ミューテックス
  - データキュー
  - 優先度データキュー
  - メールボックス ← HRP2カーネルではサポートしない
  - メッセージバッファ ← 拡張パッケージでサポート（EV3RTには未導入）
- メモリプール管理機能
  - 固定長メモリプール

- 時間管理機能
  - システム時刻管理
  - 周期ハンドラ
  - アラームハンドラ
  - オーバランハンドラ
- システム状態管理機能
- メモリオブジェクト管理機能
- 割込み管理機能
  - 割込みサービスルーチン
- CPU例外管理機能
- 拡張サービスコール管理機能
- システム構成管理機能

タスク管理機能

- タスクの状態を直接的に操作するための機能

タスク管理機能のAPI

“*”は，拡張パッケージでサポート

CRE_TSK, acre_tsk*	タスクの生成
SAC_TSK, sac_tsk*	タスクのアクセス許可ベクタの設定
del_tsk*	タスクの削除
act_tsk, iact_tsk	タスクの起動
can_act	タスク起動要求のキャンセル
ext_tsk	自タスクの終了
ter_tsk	タスクの強制終了
chg_pri	タスクの優先度の変更
get_pri	タスクの優先度の参照
get_inf	自タスクの拡張情報の参照
ref_tsk	タスクの状態参照

- タスク起動要求のキューイングとその無効化機能を持つ

タスク付属同期機能

- タスクを直接的に操作して同期を実現するための機能

タスク付属同期機能のAPI

slp_tsk, tslp_tsk	起床待ち
wup_tsk, iwup_tsk	タスクの起床
can_wup	タスク起床要求のキャンセル
rel_wai, irel_wai	強制的な待ち解除
sus_tsk	強制待ち状態への移行
rsm_tsk	強制待ち状態からの再開
dis_wai, idis_wai	待ち禁止状態への遷移
ena_wai, iena_wai	待ち禁止状態の解除
dly_tsk	自タスクの遅延

- タスク起床要求のキューイングとその無効化機能を持つ
- タイムアウト付きの起床待ちと自タスクの遅延は，動作は似ているが，どちらが正常系であるかが違う

タスク例外機能

- タスクに対する割込み/例外に対応（仮想化された割込み）
- 明示的なサービスコール呼出しにより，タスクに例外を発生させる
- 次にそのタスクが実行されるタイミング（厳密には，他にも条件あり）で，タスク例外処理ルーチンが呼び出される
- UNIXのシグナル機能/シグナルハンドラに対応

タスク例外機能のAPI

“*”は，拡張パッケージでサポート

DEF_TEX, def_tex*	タスク例外処理ルーチンの定義
ras_tex, iras_tex	タスク例外処理の要求
dis_tex	タスク例外処理の禁止
ena_tex	タスク例外処理の許可
sns_tex	タスク例外処理禁止状態の参照
ref_tex	タスク例外処理の状態参照

タスク間同期・通信機能

- HRP2カーネルでは，タスク間同期・通信のために以下の機能を用意

（主に）共有メモリによる通信のための機能

- セマフォ（排他制御，事象通知）
- イベントフラグ（事象通知）
- ミューテックス（排他制御）

メッセージ通信のための機能

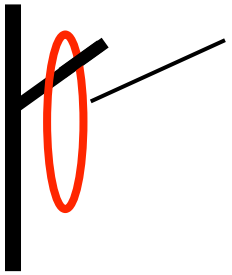
- データキュー（同期・非同期，1ワードのメッセージ）
- 優先度データキュー（同期・非同期，1ワードのメッセージ，優先度順配送）
- メッセージバッファ（同期・非同期，可変長メッセージ）

拡張パッケージでサポート（EV3RTには未導入）

セマフォ機能のAPI

“*”は，拡張パッケージでサポート

- 使用されていない資源の有無や数量を数値（セマフォの資源数）で管理することにより排他制御を実現
- セマフォは「信号」の意味



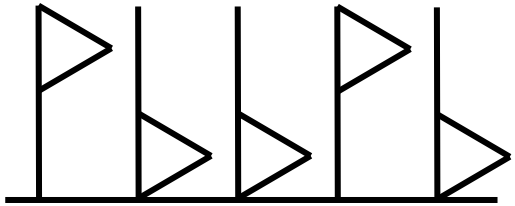
- 列車が単線区間に入る場合には，この輪（これがセマフォ資源と考えるとわかりやすい）を取る
- 一般には輪は複数あっても良い

CRE_SEM，acre_sem*	セマフォの生成
SAC_SEM，sac_sem*	セマフォのアクセス許可ベクタの設定
del_sem*	セマフォの削除
sig_sem，isig_sem	セマフォ資源の返却
wai_sem，pwai_sem，twai_sem	セマフォ資源の獲得
ini_sem	セマフォの再初期化
ref_sem	セマフォの状態参照

イベントフラグ機能のAPI

“*”は，拡張パッケージでサポート

- イベントフラグは，タスク間でイベントの発生を伝えることで同期するための機構．1つのイベントフラグは，複数のフラグで構成



CRE_FLG，acre_flg*	イベントフラグの生成
SAC_FLG，sac_flg*	イベントフラグのアクセス許可ベクタの設定
del_flg*	イベントフラグの削除
set_flg，iset_flg	イベントフラグのセット
cls_flg	イベントフラグのクリア
wai_flg，pol_flg，twai_flg	イベントフラグ待ち
ini_flg	イベントフラグの再初期化
ref_flg	イベントフラグの状態参照

- wai_flgは，指定したフラグのすべてがセットされたか，いずれかがセットされたかの条件で待ち合わせ可能

データキュー機能のAPI

“*”は，拡張パッケージでサポート

- 1ワードメッセージの非同期メッセージ通信機構
  - リングバッファで実現することを想定
- メッセージは整数としてもポインタとしてもよい
- メッセージが無い/フルの時に待ち合わせる機能
- データキュー領域のサイズを0に設定すると，同期メッセージ通信も実現可能

CRE_DTQ, acre_dtq*	データキューの生成
SAC_DTQ, sac_dtq*	データキューのアクセス許可ベクタの設定
del_dtq*	データキューの削除
snd_dtq, psnd_dtq, tsnd_dtq, ipsnd_dtq	データキューへの送信
fsnd_dtq, ifsnd_dtq	データキューへの強制送信
rcv_dtq, prcv_dtq, trcv_dtq	データキューからの受信
ini_dtq	データキューの再初期化
ref_dtq	データキューの状態参照

優先度データキュー機能のAPI

“*”は，拡張パッケージでサポート

- 1ワードメッセージの非同期メッセージ通信機構
- データを送信する時に，メッセージの優先度も渡す．メッセージは優先度順にキューイングされる
- データを受信する時に，メッセージの優先度も受け取れる
- メッセージは整数としてもポインタとしてもよい
- メッセージが無い/フルの時に待ち合わせる機能

CRE_PDQ，acre_pdq*	優先度データキューの生成
SAC_PDQ，sac_pdq*	優先度データキューのアクセス許可ベクタの設定
del_pdq*	優先度データキューの削除
snd_pdq，psnd_pdq，tsnd_pdq，ipsnd_pdq	優先度データキューへの送信
rcv_pdq，prcv_pdq，trcv_pdq	優先度データキューからの受信
ini_pdq	優先度データキューの再初期化
ref_pdq	優先度データキューの状態参照

優先度データキュー導入の理由

- μITRON4.0仕様では，メールボックスに送信するメッセージの先頭の領域（数バイト）をカーネルが利用
- アプリケーションが誤ってこの領域を書き換えると，カーネルの中で不具合が発生する
- μITRON4.0仕様 保護機能拡張（PX仕様）では，カーネルの用いる管理領域を分離するようメールボックスの仕様を変更．ただし，元の仕様のメールボックスで発生しない送信時のメッセージフルエラーが発生する

↓

- PX仕様のメールボックス機能の上位互換となる優先度データキュー機能（データを優先度順にキューイングするデータキュー）を新たに追加
- メールボックスは仕様変更せずに残し，使用は推奨しない
  - HRP2カーネルではサポートしていない

ミューテックス機能のAPI

“*”は，拡張パッケージでサポート

- 優先度上限プロトコルをサポートする排他制御機構
  → POSIXリアルタイム拡張のミューテックスに相当
- 上限のない優先度逆転を防止
- 最大資源数が1のセマフォとの他の違い
  - ロックしたタスク以外はロック解除できない
  - タスク終了時に自動的にロック解除される

CRE_MTX, acre_mtx*	ミューテックスの生成
SAC_MTX, sac_mtx*	ミューテックスのアクセス許可ベクタの設定
del_mtx*	ミューテックスの削除
loc_mtx, ploc_mtx, tloc_mtx	ミューテックスのロック
unl_mtx	ミューテックスのロック解除
ini_mtx	ミューテックスの再初期化
ref_mtx	ミューテックスの状態参照

メモリ管理機能

サポートする/しないメモリ管理機能

- 多重メモリ空間はサポートしない
  - MMUを使うことによるオーバヘッドが大きく，組込みシステム向けRTOSではメリットが小さい
- システム共通のメモリ領域をタスクに割り当てる機能（C言語のmalloc/freeと類似の機能）をサポート

メモリプール管理機能

- システム共通のメモリ領域を複数のメモリプールに分割
  - 目的によってメモリプールを分けると，重要な処理にメモリを残しておくといった使い方が可能
- HRP2カーネルでは，獲得できるメモリブロックのサイズが固定の固定長メモリプール機能のみをサポート

固定長メモリプール機能のAPI

“*”は，拡張パッケージでサポート

- 固定長のメモリブロックを獲得/返却するための機能
- メモリブロックのサイズと個数は，メモリプールの生成時に静的に与える
- メモリが足りない時に待ちに入る機能
- メモリフラグメンテーション（断片化）が起こらない
  - 可変長メモリプールは，便利であるが，メモリフラグメンテーションが起こるため，メモリ不足時の対処が難しい

CRE_MPF，acre_mpf*	固定長メモリプールの生成
SAC_MPF，sac_mpf*	固定長メモリプールのアクセス許可ベクタの設定
del_mpf*	固定長メモリプールの削除
get_mpf, pget_mpf, tget_mpf	固定長メモリブロックの獲得
rel_mpf	固定長メモリブロックの返却
ini_mpf	固定長メモリプールの再初期化
ref_mpf	固定長メモリプールの状態参照

時間管理機能

システム時刻管理，性能評価用システム時刻管理

- システム時刻（カーネルが管理する時刻）を管理する機能

タイムイベントハンドラ

- 時間の経過をきっかけに呼び出されるハンドラ
- HRP2カーネルでは次の3種類のタイムイベントハンドラをサポート
  - 周期ハンドラ（周期的に呼ばれる）
  - アラームハンドラ（指定した時間後に呼ばれる）
  - オーバランハンドラ

時間同期のためのその他の機能

- 自タスクの遅延（タスクを指定時間待たせる）
- 待ちに入るシステムコールのタイムアウト機能

システム時刻管理機能のAPI

get_tim	システム時刻の参照

性能評価用システム時刻管理機能のAPI

get_utm	性能評価用システム時刻の参照

周期ハンドラ機能のAPI

“*”は，拡張パッケージでサポート

- 指定した周期で起動されるタイムイベントハンドラ
- 周期ハンドラの先頭番地と起動周期は，周期ハンドラの生成時に静的に設定

CRE_CYC，acre_cyc*	周期ハンドラの生成
SAC_CYC，sac_cyc*	周期ハンドラのアクセス許可ベクタの設定
del_cyc*	周期ハンドラの削除
sta_cyc	周期ハンドラの動作開始
stp_cyc	周期ハンドラの動作停止
ref_cyc	周期ハンドラの状態参照

アラームハンドラ機能のAPI

“*”は，拡張パッケージでサポート

- 指定した相対時間後に起動されるタイムイベントハンドラ
- アラームハンドラの先頭番地は，アラームハンドラの生成時に静的に設定
- アラームハンドラの起動時刻は，sta_alm/ista_almのパラメータで相対時間により指定

CRE_ALM，cre_alm*	アラームハンドラの生成
SAC_ALM，sac_alm*	アラームハンドラのアクセス許可ベクタの設定
del_alm*	アラームハンドラの削除
sta_alm，ista_alm	アラームハンドラの動作開始
stp_alm，istp_alm	アラームハンドラの動作停止
ref_alm	アラームハンドラの状態参照

オーバランハンドラ機能のAPI

“**”は，HRP2のサポート外

- タスクが使用したプロセッサ時間（経過時間とは異なる）が，指定した時間を超えた場合に起動されるタイムイベントハンドラ
- オーバランハンドラはすべてのタスクに共通で，その先頭番地はオーバランハンドラの定義時に静的に設定
- タスクが使用できるプロセッサ時間は，sta_ovr/ista ovrのパラメータで指定

DEF_OVR，def_ovr**	オーバランハンドラの定義
sta_ovr，ista_ovr	オーバランハンドラの動作開始
stp_ovr，istp_ovr	オーバランハンドラの動作停止
ref_ovr	オーバランハンドラの状態参照

システム状態管理機能

- システム状態を参照/変更するための機能

システム状態管理機能のAPI

rot_rdq, irot_rdq	タスクの優先順位の回転
get_tid, iget_tid	実行状態のタスクIDの参照
get_did	実行状態のタスクが属する保護ドメインIDの参照
loc_cpu, iloc_cpu	CPUロック状態への遷移
unl_cpu, iunl_cpu	CPUロック状態の解除
dis_dsp	ディスパッチの禁止
ena_dsp	ディスパッチの許可
sns_ctx	コンテキストの参照
sns_loc	CPUロック状態の参照
sns_dsp	ディスパッチ禁止状態の参照
sns_dpn	ディスパッチ保留状態の参照
sns_ker	カーネル非動作状態の参照
ext_ker	カーネルの終了

メモリオブジェクト管理機能

- メモリオブジェクトの配置を決定し，状態を参照/変更するための機能

メモリオブジェクト管理機能のAPI

“**”は，HRP2のサポート外

ATT_REG	メモリリージョンの登録
DEF_SRG	標準メモリリージョンの定義
ATT_SEC, ATA_SEC	セクションの登録
LNK_SEC	セクションの配置
ATT_MOD, ATA_MOD	オブジェクトモジュールの登録
ATT_MEM, ATA_MEM, att_mem**	メモリオブジェクトの登録
ATT_PMA, ATA_PMA, att_pma**	物理メモリ領域の登録
sac_mem**	メモリオブジェクトのアクセス許可ベクタの設定
det_mem**	メモリオブジェクトの登録解除
prb_mem	メモリ領域に対するアクセス権のチェック

割込み処理モデルと割込み管理機能

割込み処理の流れ（概要）

- 割込みが発生すると，カーネル内の出入口処理を経由して，アプリケーションが登録した割込みサービスルーチン（ISR）または割込みハンドラが呼び出される
- ISR/割込みハンドラはタスクよりも優先して実行
- ISR/割込みハンドラの中でサービスコールを呼び出して，タスクの起動/起床などが可能
- アプリケーションは，プロセッサの割込みアーキテクチャに依存せずに記述できるISRを用意するのが基本
- 割込みハンドラはコンフィギュレータによって生成されるのが基本だが，特殊なケースに対応するために，ユーザ側で用意することもできる

割込み管理機能のAPI

“*”は，拡張パッケージでサポート
“**”は，HRP2のサポート外

- 割込み要求ライン属性の設定（CFG_INT）により，割込み要求ラインの優先度，レベルトリガかエッジトリガか，初期状態で割込みをマスクするか，などを設定できる
- 割込みの禁止（dis_int）と許可（ena_int）は，割込み番号を指定して，その割込みのみをマスク/マスク解除

CFG_INT，cfg_int**	割込み要求ラインの属性の設定
ATT_ISR，CRE_ISR**，acre_isr*	割込みサービスルーチンの生成/追加
SAC_ISR**，sac_isr*	割込みサービスルーチンのアクセス許可ベクタの設定
del_isr*	割込みサービスルーチンの削除
DEF_INH，def_inh**	割込みハンドラの定義
dis_int	割込みの禁止
ena_int	割込みの許可
chg_ipm	割込み優先度マスクの変更
get_ipm	割込み優先度マスクの参照

CPU例外処理モデルとCPU例外管理機能

CPU例外処理の流れ

- CPU例外が発生すると，カーネル内の出入口処理を経由して，アプリケーションが登録したCPU例外ハンドラが呼び出される
- CPU例外ハンドラはタスクよりも優先して実行
- CPU例外ハンドラの中でサービスコールを呼び出して，タスクの起動/起床やタスクに対して例外を発生させることなどが可能
- CPU例外ハンドラを，プロセッサのアーキテクチャに依存せずに記述することは（現時点では）不可能
- CPU例外ハンドラは，CPU例外ハンドラ番号を指定してカーネルに登録

CPU例外からのリカバリのアプローチ

(1) カーネルに依存せずに，CPU例外の原因を取り除き，実行を継続する

(2) CPU例外を起こしたタスクより優先度の高いタスクを起動/起床して，その中でリカバリ処理を行う

(3) CPU例外を起こしたタスクにタスク例外処理を要求し，その中でリカバリ処理を行う

(4) システム全体に対してリカバリ処理を行う(例えば，システムを再起動する)

CPU例外管理機能のAPI

“**”は，HRP2のサポート外

- CPU例外ハンドラの中で，どのリカバリアプローチが採れるかを判断するためのAPIを用意

DEF_EXC，def_exc**	CPU例外ハンドラの定義
xsns_dpn	CPU例外発生時のディスパッチ保留状態の参照
xsns_xpn	CPU例外発生時のタスク例外処理保留状態の参照

拡張サービスコール管理機能

拡張サービスコールとは？

- 非特権モードで実行される処理単位から，特権モードで実行すべきルーチンを呼び出すための機能（特権モードで実行される処理単位からも呼び出すことができる）
  - 拡張サービスコールはカーネルドメインに属する
  - 拡張サービスコールからは，すべてのカーネルオブジェクトに対して，すべての種別のアクセスを行える
- システムサービス（ミドルウェアやデバイスドライバ）の呼び出しに使用することを想定した機能

拡張サービスコール管理機能のAPI

“**”は，HRP2のサポート外

DEF_SVC, def_svc**	拡張サービスコールの定義

システム構成管理機能

“#”は，拡張パッケージでサポート
（統合仕様書には未記載）

システム構成管理機能のAPI

- LMT_DOMにより，ユーザドメインで使用できるタスクの優先度を制限することができる
- DEF_ICSとDEF_KMMにより，非タスクコンテキスト用のスタック領域とカーネルが割り付けるメモリ領域（管理領域等を自動割り付けにした場合に使われる）を設定できる
- 初期化ルーチンは，カーネルの初期化時，カーネルの動作開始前に呼ばれる
- 終了処理ルーチンは，カーネルの動作終了後に呼ばれる

LMT_DOM	保護ドメインに対する制限の設定
DEF_ICS	非タスクコンテキスト用スタック領域の設定
DEF_KMM#	カーネルが割り付けるメモリ領域の設定
ATT_INI	初期化ルーチンの追加
ATT_TER	終了処理ルーチンの追加