µT-Kernel 入門

2013年 12月 18-19日（水，木）

【T-Engine学術・教育WG】

スパンション　長濱　美保

旧 富士通セミコンダクター

目次

第1章

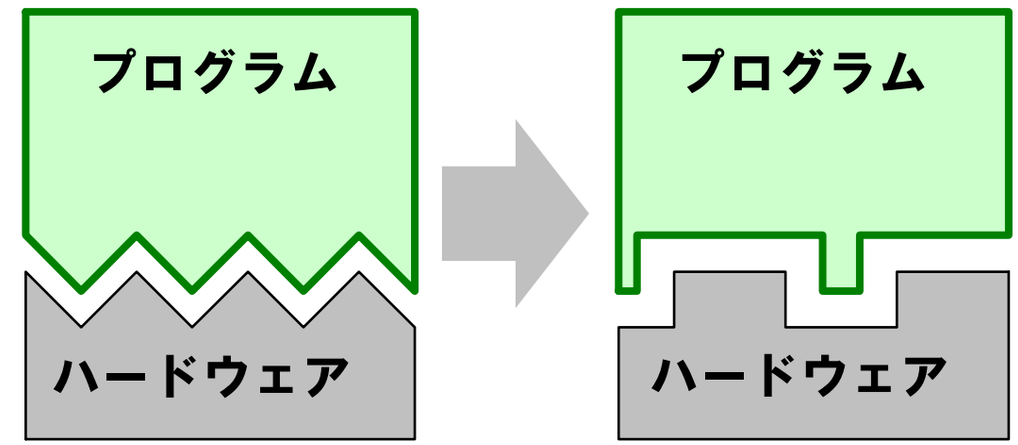
µT-Kernel 概要

OS利用のメリット（1）

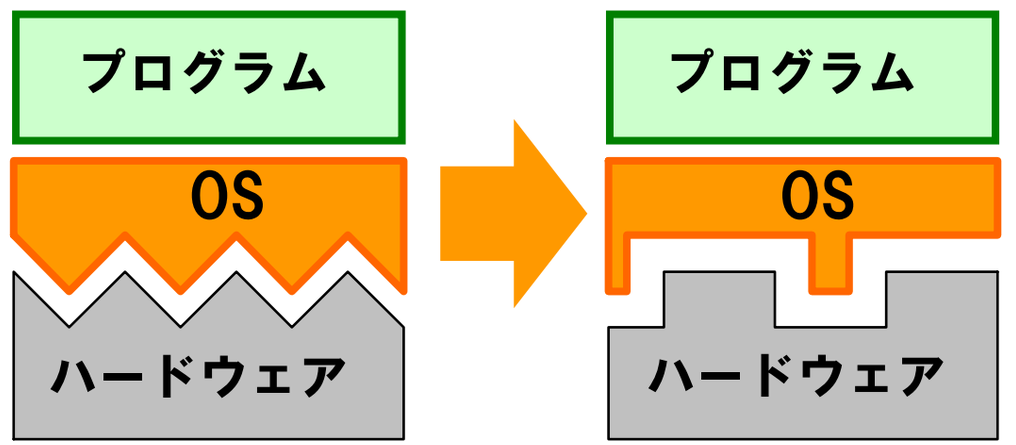
I. プログラムの再利用性が高い

OS使用以前
●ハードウェアに依存する箇所は全て作り直し

OS使用後
●OSでハードウェア依存部を吸収するため、
プログラムの再利用性が高くなる



再利用性
低い



再利用性
高い

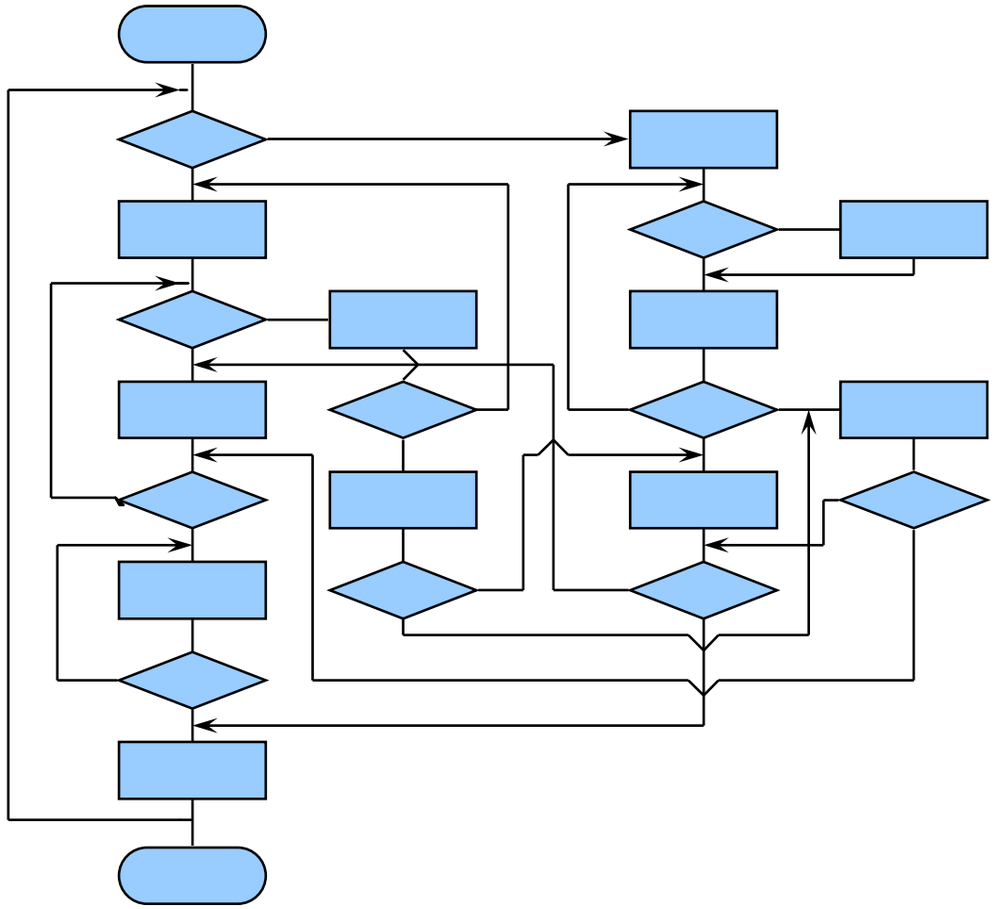
OS利用のメリット（2）

すっきりとした見やすいプログラムになる

OS使用以前

●メンテナンスが困難

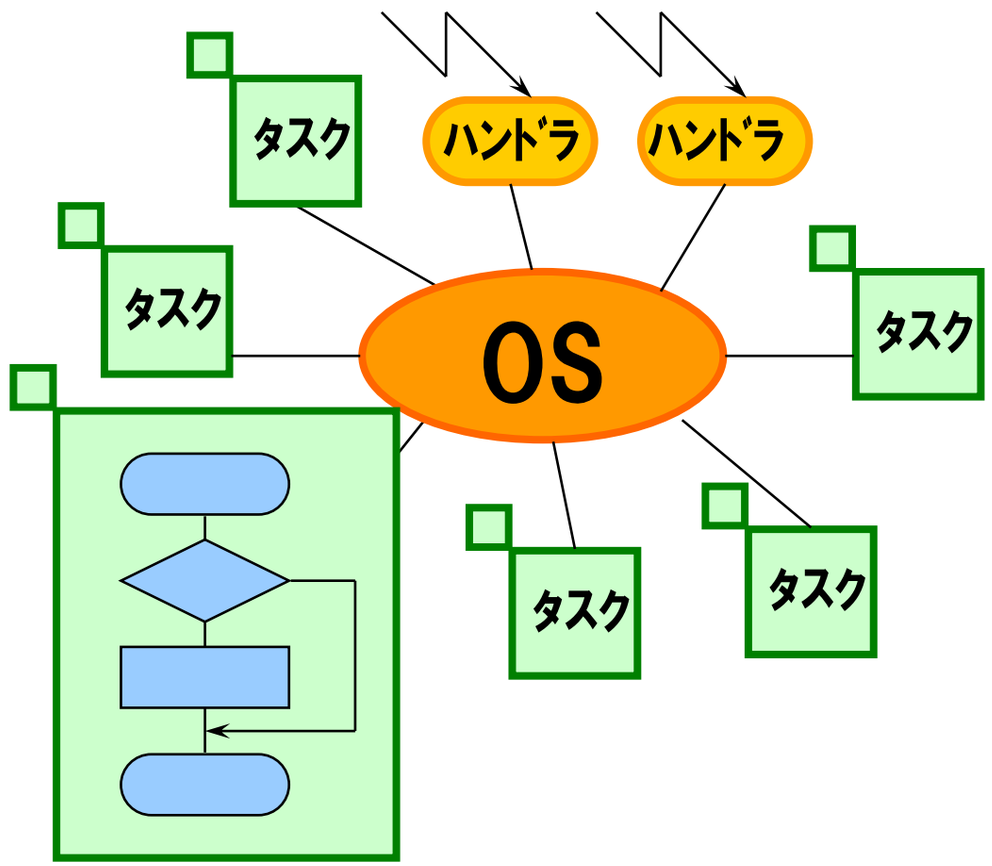
●簡単な機能追加するだけでも、検証作業大



OS使用後

●タスク単位でプログラムが閉じているため、メンテナンスが容易

●新しく追加したタスク（機能）について、検証作業を集中することが可能



OSに任せると便利になること

1. 同期（排他）制御を独自に作りこまなくて良い
2. 周期ハンドラ，スリープなど時間管理が使える
3. 割込みハンドラの登録などもシステムコール利用可

【周期ハンドラ未使用】

```
write_mail(){
          :
  transmit_position();
          :
  transmit_position();
          :
  transmit_position();
          :
  transmit_position();
}
```

10ms毎
送信プログラム

【周期ハンドラ使用】

```
Initial_tsk(){
  tk_cre_cyc(trans_posi_10ms);
}
```

10ms毎

```
transmit_position(){
  read_sensor_value();
}
```

```
write_mail(){
        :
        :
        :
        :
}
```

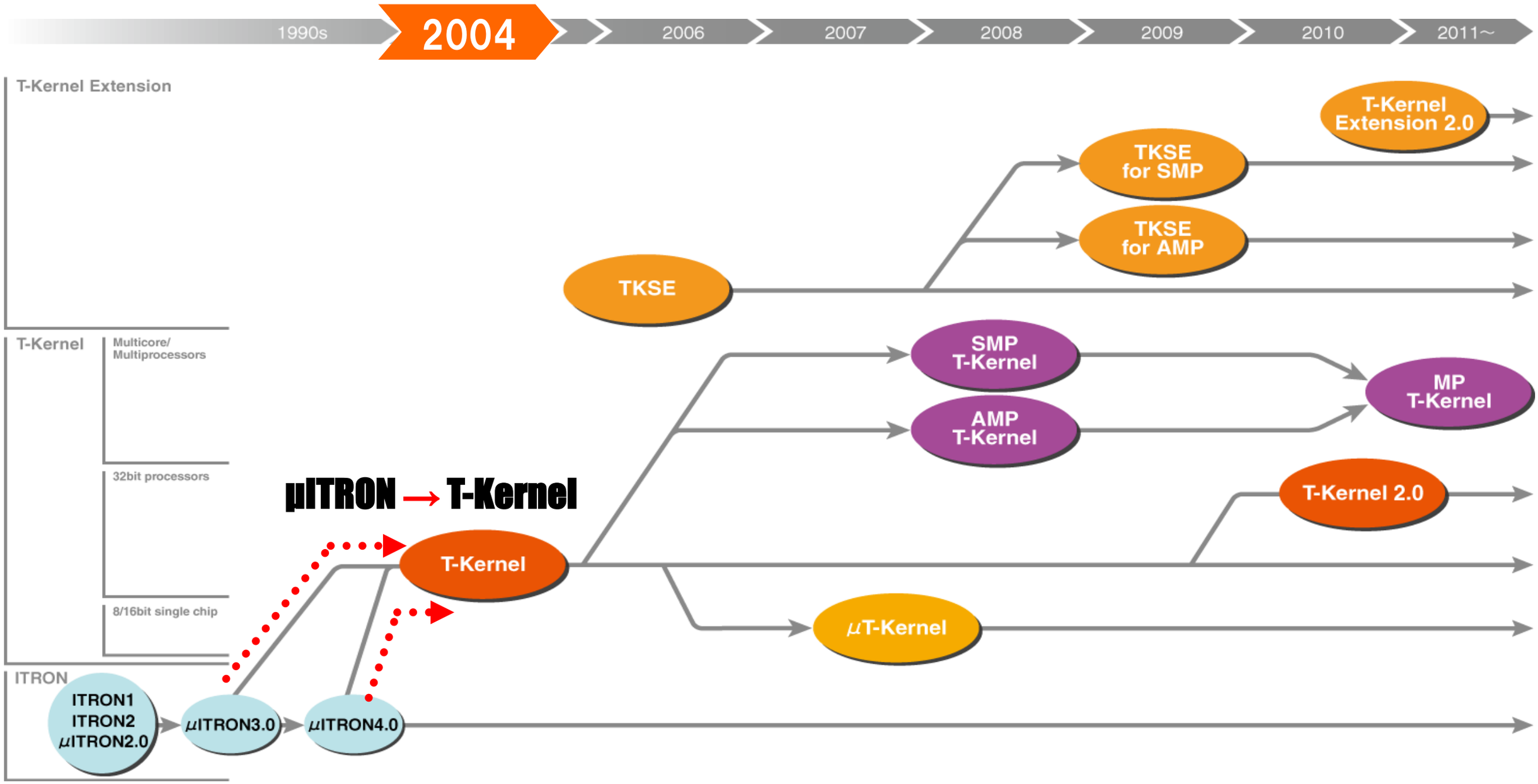
排他処理の実現

並行性の実現手段	タスク	割込み処理
排他制御の手段	セマフォが代表 他にもRTOSの提供する多くの手段がある	割込み禁止および許可
排他制御の(悪)影響	関連するタスクのみに限られる 関連しない優先度の高いタスクには影響を与えない 優先度逆転などのデッドロックに注意	システム全体，全ISRに及ぶ優先度の高いISRを停止させてしまう
特徴	多項目に及ぶが使い方を誤ってもシステム全体に影響を及ぼすことはまれ ただし，優先度の高いタスクで使用する場合は要注意 また，デッドロックにも要注意	比較的単純に使えるが，使い方を誤るとシステム全体の性能に影響を及ぼすため危険 使い方を誤ると最悪システムダウンを招く

OS導入により，ソフトウェアの生産性，再利用性が向上！！

µT-Kernel とは？

T-Kernel 仕様の誕生（1）



T-Kernel 仕様の誕生 (2)

ITRON による組込み機器ソフトウェア開発の限界

・他社ITRON間でのアプリの移植が難しい
・高機能OSへの移行が難しい

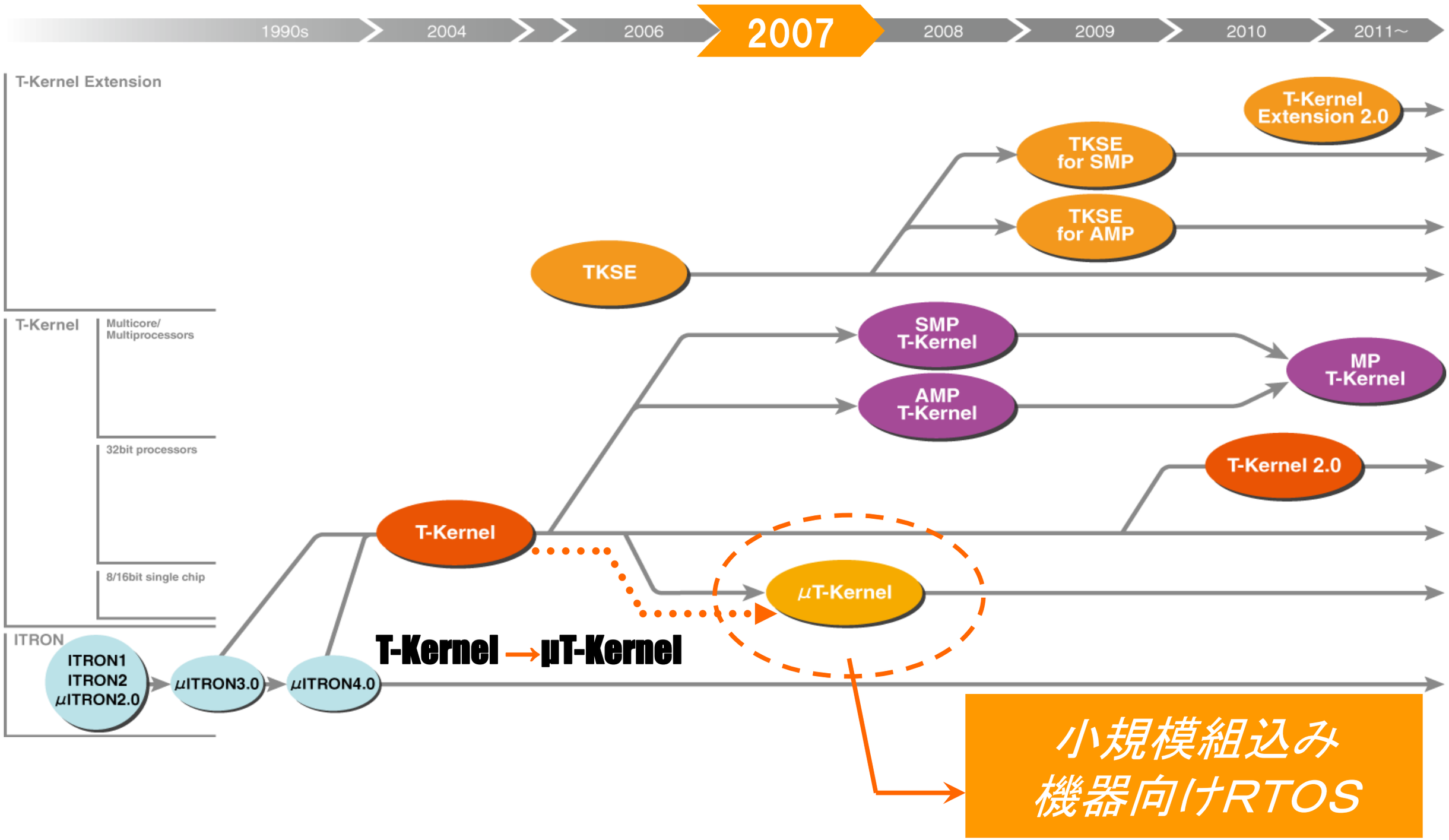
セットメーカ、半導体メーカ、OSベンダが、次世代組込みOSとして「T-Kernel仕様」の策定、開発を推進

■トロンプロジェクトの20年の成果を踏襲

■新たな要求に対応

ソフトウェアの再利用	強い標準化
高機能なソフトウェア開発	ファイルシステム、デバイス管理機能、省電力機能、ダイナミックロード機能
信頼性の高いソフトウェア開発	プロセス単位のプログラム管理機能
より高度なソフトウェア開発	OSの拡張機能

T-Kernel 仕様からμT-Kernel 仕様へ（1）



T-Kernel 仕様からμT-Kernel 仕様へ（2）

T-Kernelの小規模組込み機器への適用の要求

・T-Kernelは大規模な組込みシステムを視野に入れているため小規模な組込みシステムでは使わない機能があり、フットプリント、オーバヘッドが大きい

T-Engineフォーラムにおいて、小規模組込み機器へ適用できるOSとして「μT-Kernel 仕様」を策定

■シングルチップマイコン、8/16bitマイコンへの適用

■T-Kernelとの互換性を考慮

最適化・適応化 → 小規模システムでは使われない機能の削除
システム全体のオーバヘッドになる機能の削除
資源を有効に使う機能の追加

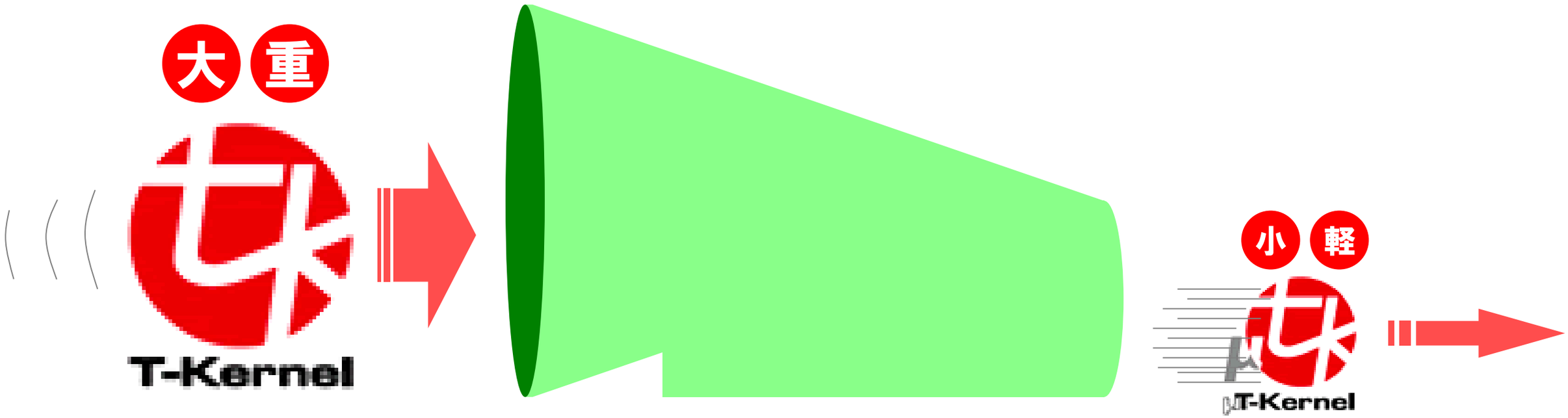
ソフトウェアの再利用 → 強い標準化、T-KernelとI/Fの統一

高機能なソフトウェア開発 → デバイス管理機能、省電力機能

µT-Kernel の特長

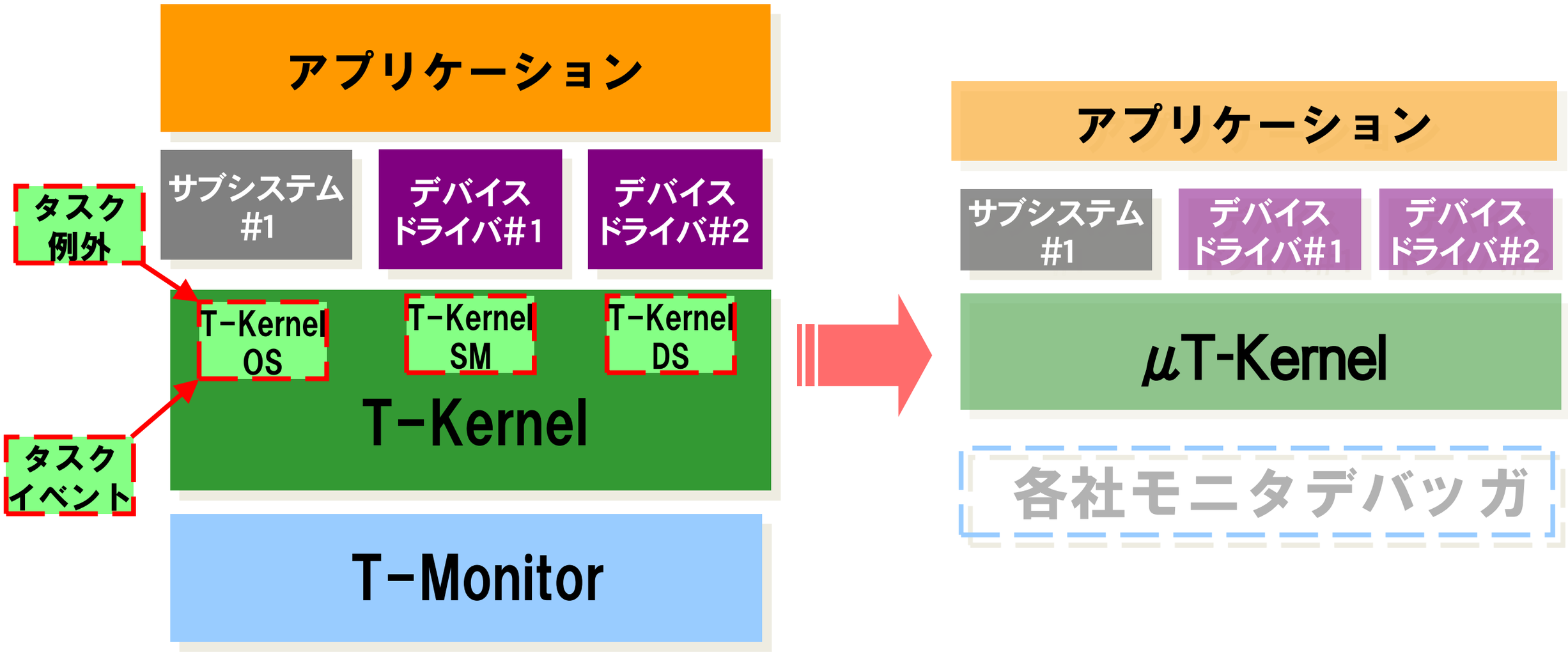
T-Kernel とμT-Kernel の差異（1）

- 必要な機能に絞ったシンプル・カーネル
- 小規模マイコン・アーキテクチャを想定した仕様
- メモリの効率化



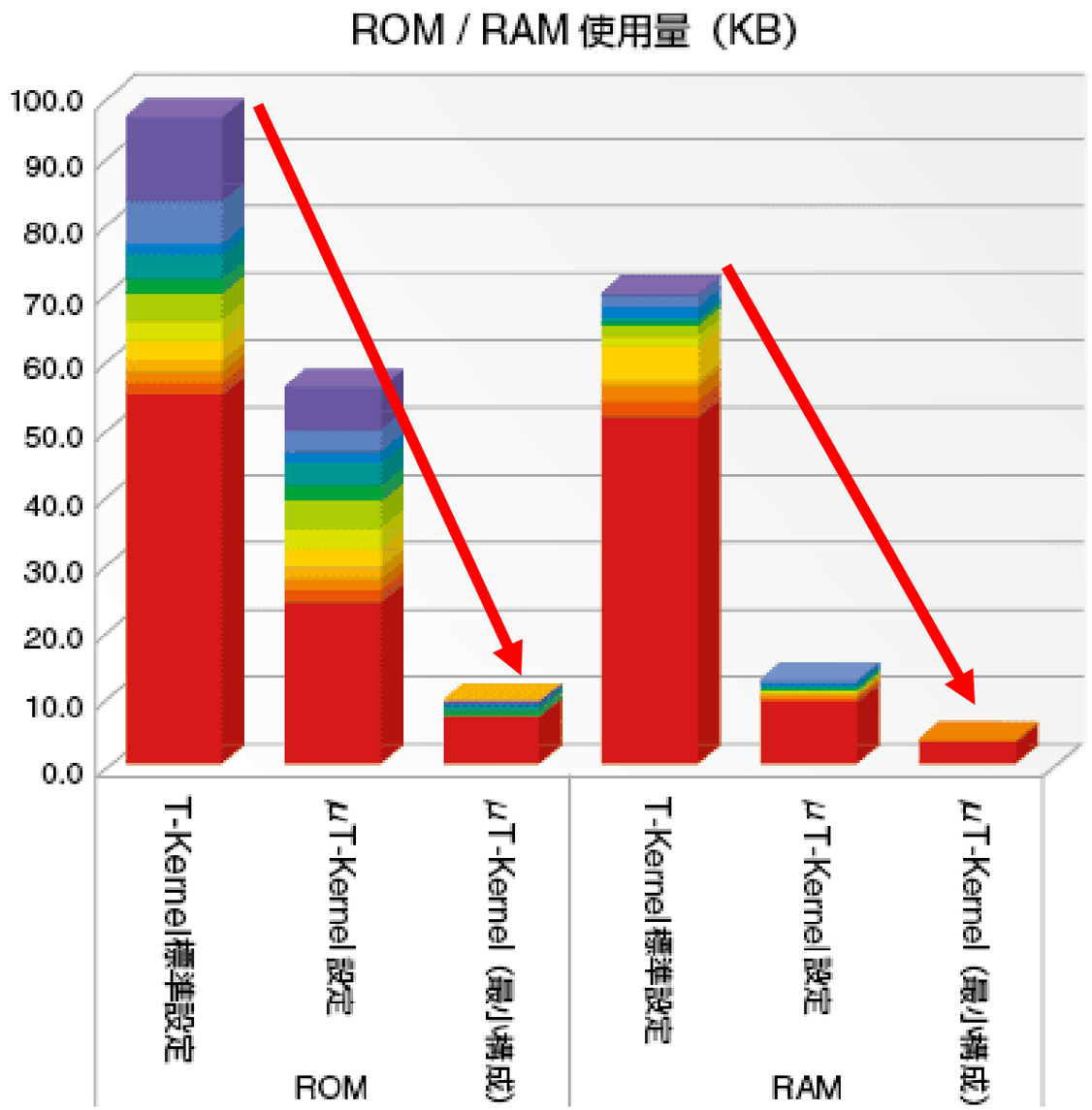
T-Kernel とμT-Kernel の差異（2）

- 必要な機能に絞ったシンプルカーネル
  - **T-Kernel/OS**, **T-Kernel/DS**, **T-Kernel/SM**の区別はなし
  - 一部機能の縮小 ・・・ タスク例外,タスクイベント等なし
  - T-Monitor は必須ではない



T-Kernel とμT-Kernel の差異（3）

- 小規模マイコンアーキテクチャを想定した仕様
  - MMU／MPU無しのシステムを想定
  - ワンチップマイコンへの対応



※ T-Engineフォーラムウェブページより

必要ROM容量：

T-Kernel ⇒ 100KB

μT-Kernel ⇒ 8KB～55KB

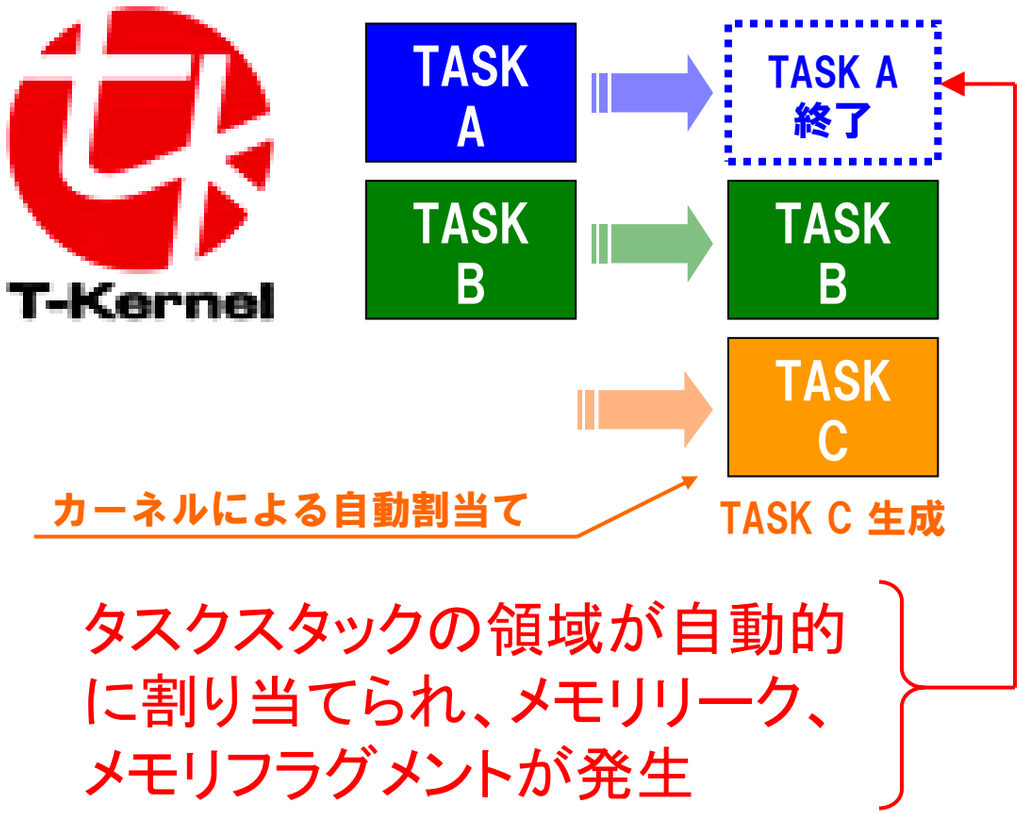
必要RAM容量：

T-Kernel ⇒ 70KB

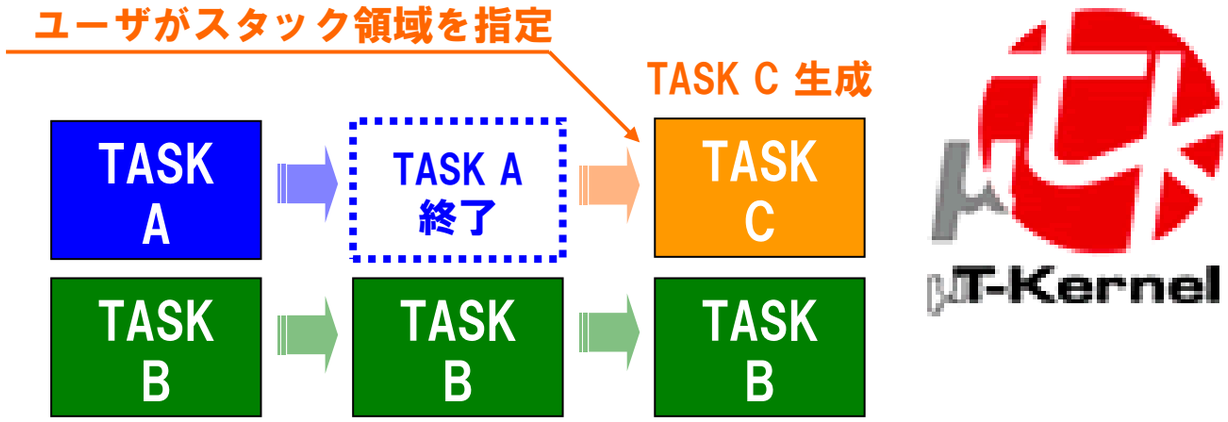
μT-Kernel ⇒ 4KB～12KB

T-Kernel とμT-Kernel の差異（4）

- メモリ管理の効率化
  - ユーザがカーネル内のメモリ獲得手段を選択できる（静的 or 動的）



タスクスタックの領域が自動的に割り当てられ、メモリリーク、メモリフラグメントが発生



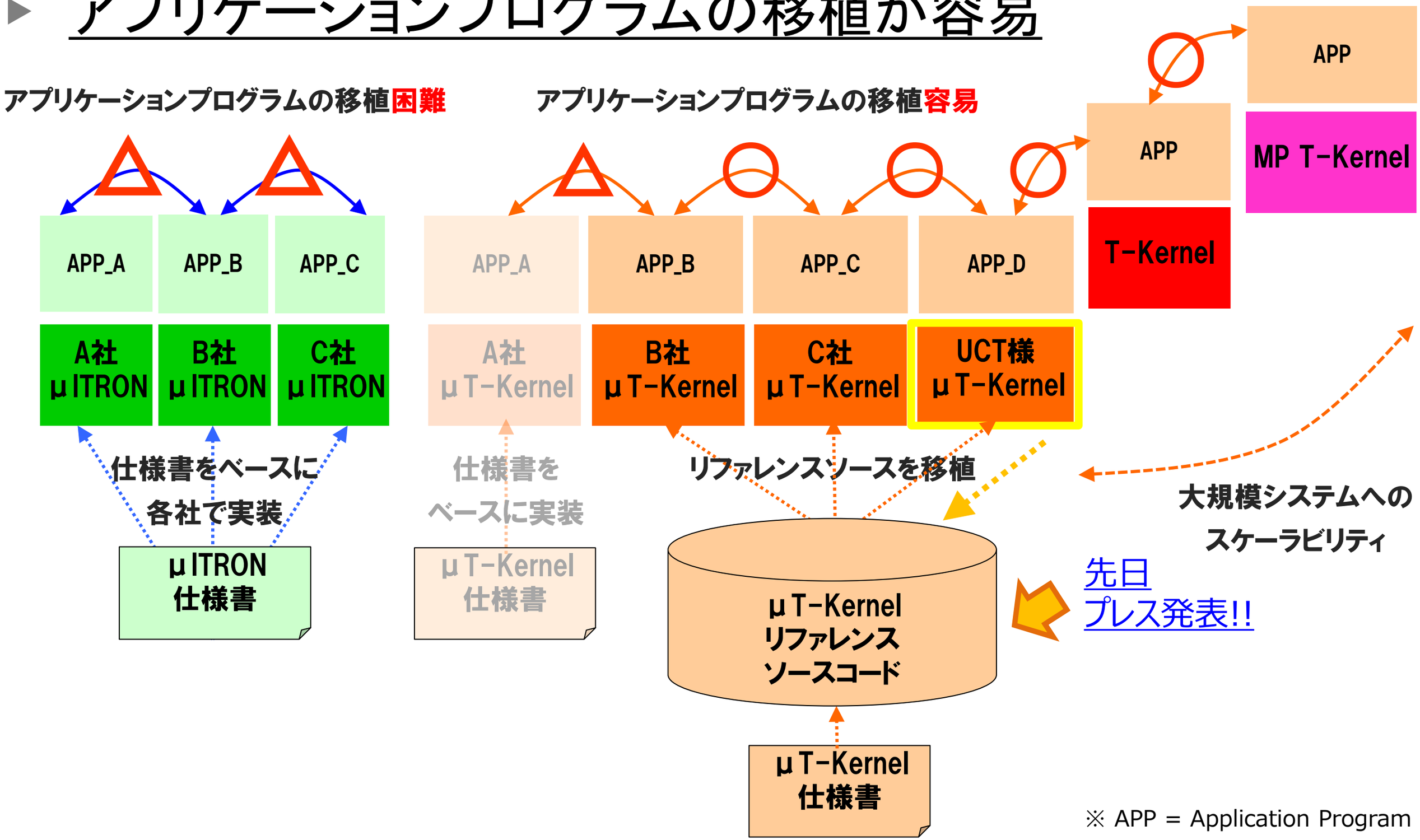
スタック領域の指定
ユーザが静的にメモリ管理を行えば、タスクスタック領域の共有が可能となり、省メモリ化を実現

メモリを静的に確保することによって、

- *メモリのフラグメント、リークを解消*
- メモリ管理機能を取り外すことで、*省メモリ化可能*

µITRON とµT-Kernel の差異（1）

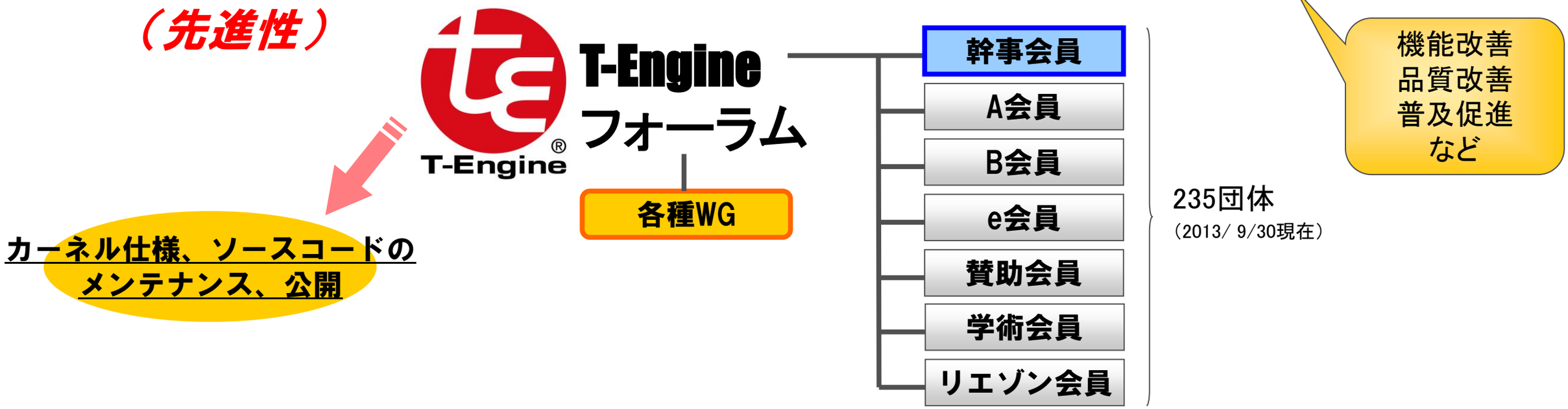
- アプリケーションプログラムの移植が容易



μITRONとμT-Kernelの差異（2）

オープンソース

- T-Engineフォーラムより、カーネル仕様とソースコードが一般公開されている
  （透明性）
- T-Engineフォーラムにおいて、カーネルソースコードの一元管理を行っている
  （一元性）
  ➡ 従来のオープンソースとは違い、第三者の著作権物が混入して
  いないことをT-Engineフォーラムが保証している。
- T-Engineフォーラムおよび会員企業で、μT-Kernel仕様を改善している
  （先進性）



µT-Kernel、T-Kernel、µITRONの相違点（1）

項目	µT-Kernel	T-Kernel	µITRON4.0
リファレンスコード	あり	あり	なし
メモリ保護機能	なし	あり	あり *1
デバイスドライバインターフェイス	あり	あり	なし
サブシステム	なし *2	あり	あり
オブジェクト生成	動的	動的	静的および動的
オブジェクト待ちの永久待ち、ポーリング、タイムアウトのシステムコール	同一システムコール	同一システムコール	別システムコール
セマフォの資源獲得/解放単位	複数	複数	1

*1「µITRON4.0仕様保護機能拡張（µITRON4.0/PX仕様）」において定義されている
*2 定義のみ。µITRON4.0の拡張SVC機能と同等

T-Kernel 2.0 Extension（T2EX）

µT-Kernel 、T-Kernel、µITRON の相違点（2）

項目	µT-Kernel	T-Kernel	µITRON4.0
タスク例外	なし	あり	あり
オーバーランハンドラ	なし	あり	あり
タスクイベント、時分割実行、 待ち禁止、システムメモリ 管理、アドレス空間管理 I/Oポートアクセス	なし	あり	あり
省電力	あり *1	あり	なし
デバッガサポート	あり	あり	なし
標準モニタデバッガ	なし *2	T-Monitor	なし

*1 「µT-Kernel 省電力機能実装ガイドライン」において定義されている
*2 各社がCPU毎に別途用意

現状の取り組み

μT-Kernel移植ガイド

- T-Engineフォーラム発行のμITRON＞μT-Kernel移植ガイド


TEF022-W001-01.00.00(ja

μITRON から
μT-Kernel/T-Kernel への
移行ガイド

Version 1.00.00
(2010/11/01)

μT-Kernel T-Kernel

T-Engine フォーラム

目次

http://www.t-engine.org/ja/wp-content/themes/wp.vicuna/pdf/specifications/ja/TEF022-W001-01.00.00_ja.pdf

μT-Licenseについて

	μT-License	T-License
ライセンス料	無償	無償
リファレンスコードの再配布	原則不可 *1	原則不可 *2
リファレンスコードの改変	可	可
改変版コード（派生物）の配布	可	不可
派生物の再改変と配布	可	—
派生物の配布規定の変更	可	—
表示義務	有り *3	有り
適合性確認	必要	—

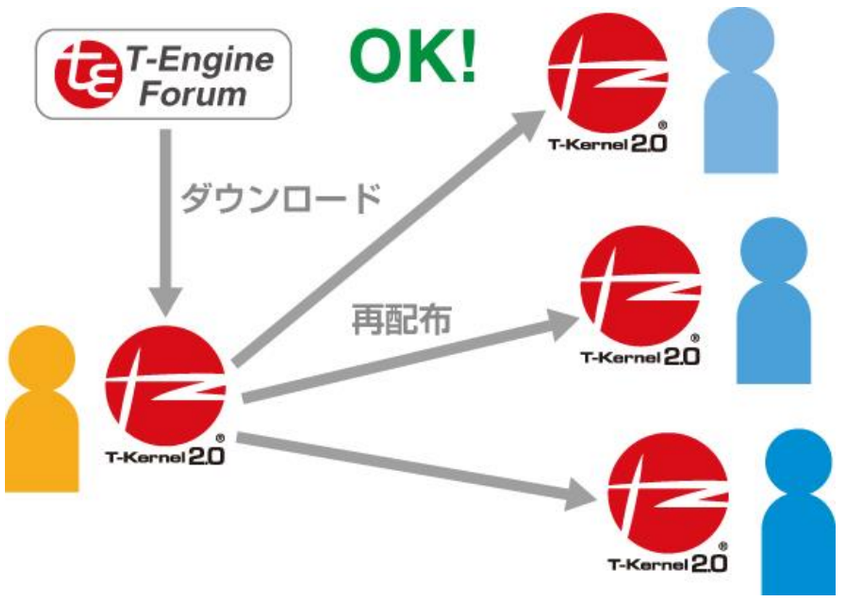
*1：以下によりオリジナルのリファレンスコードも再配布可能となる。
一切改変しないこと、μT-License を添付すること、μT-License への同意を開発者に促すこと。

*2：T-Engine フォーラムの A 会員が所定の手続きを経て承認されれば再配布可能。

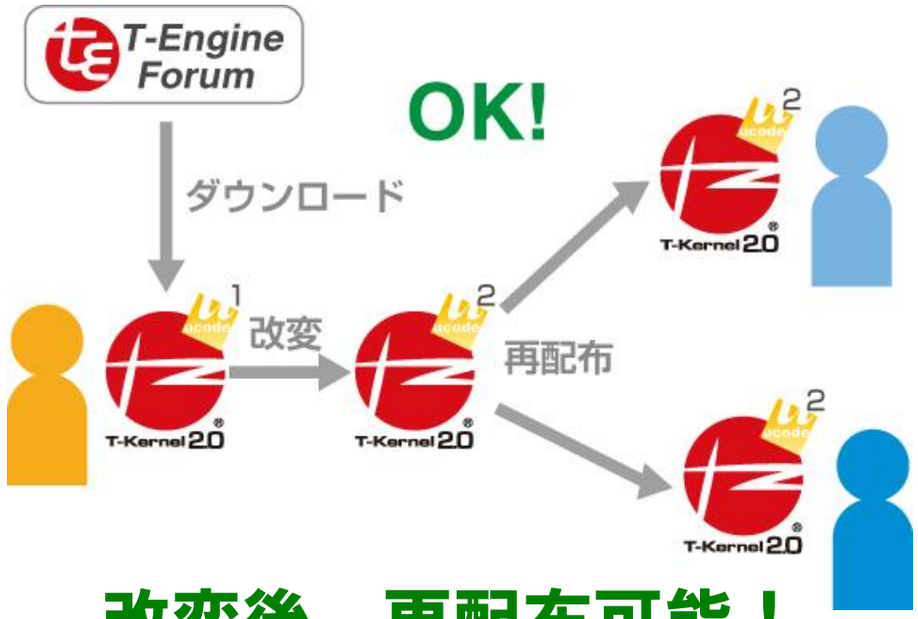
*3：T-Engine フォーラムの会員は申請により表示義務の免除を受けられる。

※ TRONWARE Vol.102 より

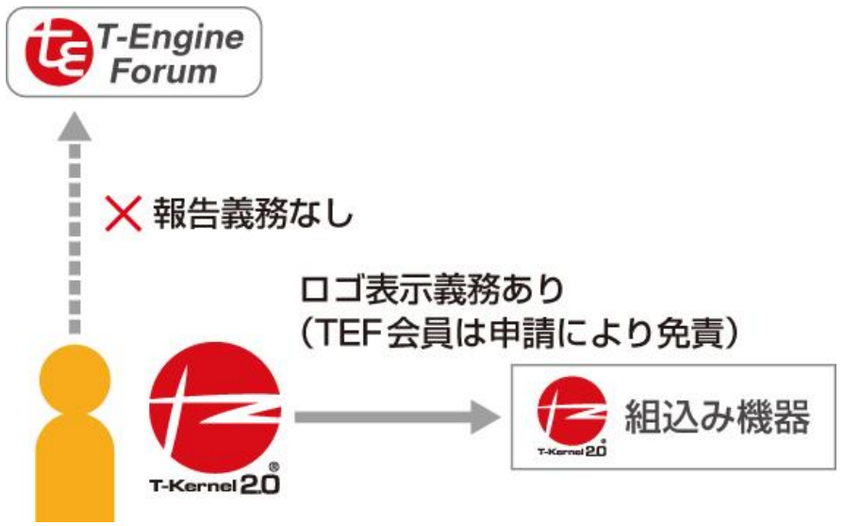
T-License 2.0について



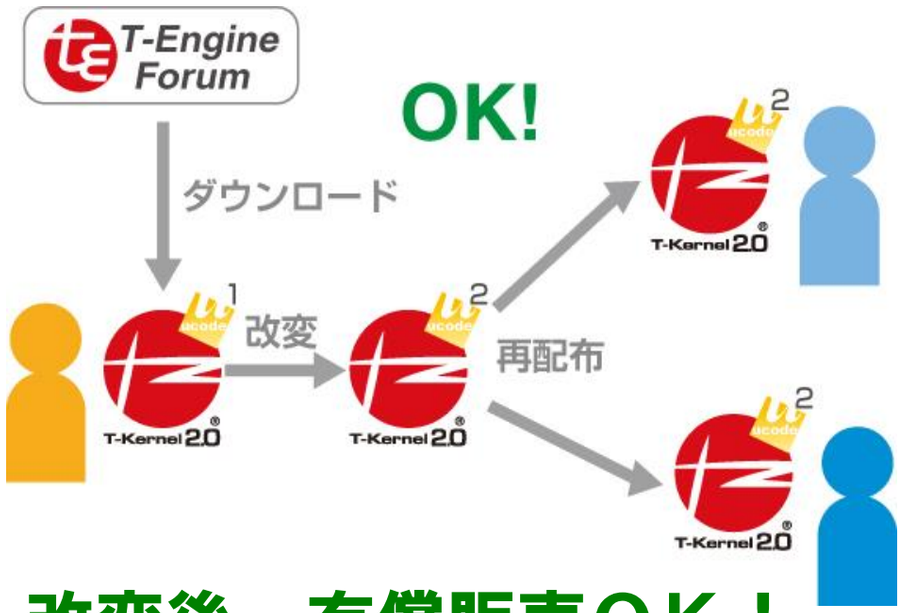
再配布可能！



改変後、再配布可能！
※ UCODE登録が必要



ロゴ表示義務あり



改変後、有償販売ＯＫ！
※ UCODE登録が必要

第2章

では実際にやってみる

μT-REALOS/M3

μT-REALOS

特長

- 多くのミドルウェアを利用できるμT-Kernel仕様に準拠
- 基本コード2.6Kbyteの小さなカーネル
- 省電力対応
- デバイス管理機能
- 必要な機能を選択できるコンフィギュレータ
- カーネル情報表示機能プラグイン
- μITRON仕様APIをサポート
- 多くの開発環境をサポート
- タスクデバッグ機能 ***new!***

製品仕様

対象CPU	FM3ファミリ
OS仕様	μT-Kernel仕様
ｺｰﾄﾞｻｲｽﾞ	2.6～30KB
開発環境	MDK-ARM（ARM KEIL） EWARM（IARｼｽﾃﾑｽﾞ） RealView Development Suite v4.x（ARM） WATCHPOINT（Sohwa & Sophia Technologies） microVIEW（横河ディジタルコンピュータ）
ｴﾐｭﾚｰﾀ	ULINK（ARM KEIL） J-Link（IARｼｽﾃﾑｽﾞ） RealView ICE（ARM） EJSCATT（Sohwa & Sophia Technologies） adviceLUNA（横河ディジタルコンピュータ）

http://www.spansion.com/JP/Support/microcontrollers/development-environment/pages/software-relatimeos-index.aspx

ソースコード

µT-KernelがサポートするCPU

CPU		開発環境	対応
型名	メーカ名		
AT91M 55800A	Atmel	GCC	ソースコード
H8S/2212	ルネサス エレクトロニクス	GCC	ソースコード
FR60 (MB91FV310A)	富士通セミコンダクター	SOFTUNE	体験版公開
M16C/M32C	ルネサス エレクトロニクス	GCC	パッチ
V850	ルネサス エレクトロニクス	GCC	パッチ
Kinetis (ARM Cortex-M4)	Freescale	IAR EWARM, ARM MDK	製品
RX62N	ルネサス エレクトロニクス	HEW	製品
STM32F2 (ARM Cortex-M3)	ST Microelectronics	IAR EWARM, ARM MDK	製品
Stellaris (ARM Cortex-M3)	Texas Instruments	IAR EWARM, ARM MDK, TI CCS	製品
TX03 (ARM Cortex-M3)	東芝セミコンダクター	IAR EWARM, ARM MDK	製品
FM3 (ARM Cortex-M3)	富士通セミコンダクター	IAR EWARM, ARM MDK	製品

ソースコードのダウンロード

http://www.t-engine.org/download/index.php?route=product/category&path=17

※ サポートCPUの詳細：

http://www.t-engine.org/ja/hwinfo#b

ソースコード

サポートCPU一覧（スパンション・イノベイツ）

CPU		開発環境	対応
型名	メーカ名		
FR60 （MB91FV310A）	富士通セミコンダクター	SOFTUNE	体験版公開
FM3 （ARM Cortex-M3）		ARM社 （MCB9BF500）	製品
		IAR社 （KSK-MB9BF506R）	
		富士通セミコンダクター （SK-FM3-48PMC-USBSTICK）	

UCT µT-Kernel DevKit tuned for FM3-GCC無償評価版、µT-Kernel Ver.1.01.02を公開しました。

- UCT µT-Kernel DevKit tuned for FM3-GCC無償評価
- µT-Kernel Ver.1.01.02

http://www.t-engine.org/ja/2013/download20130513.html

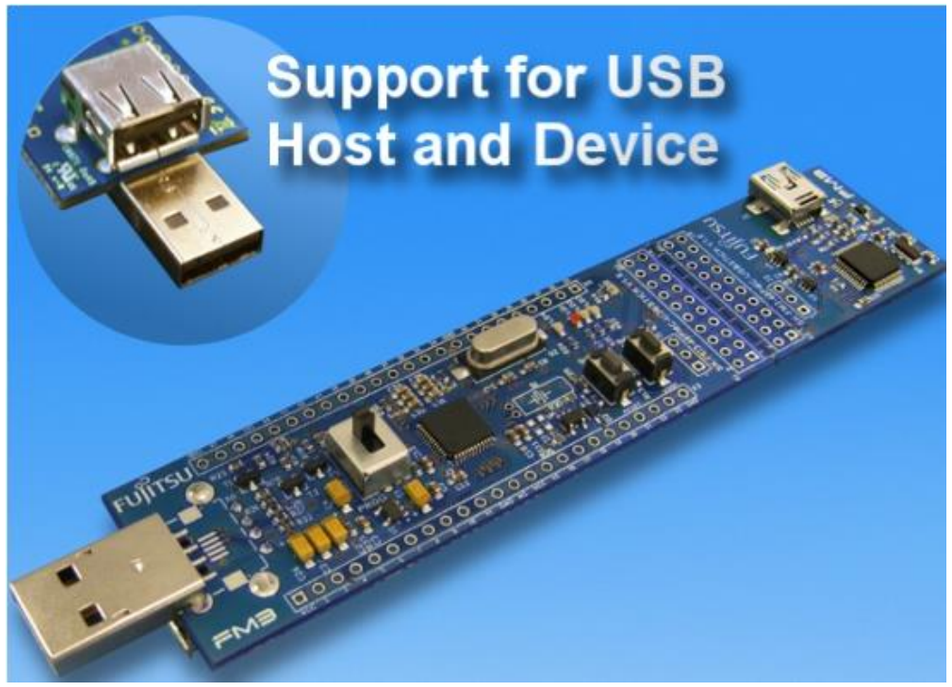
µT-Kernelは、小規模組込みシステムをターゲットとしたリアルタイムOSです。

- 1.シングルチップマイコンなどの16ビットCPUにも対応しています。
- 2.ROM/RAMが非常に少ない組込みシステムでも動作可能です。
  ※ROMが8KB、RAMが4KB程度しかない小規模な組込みシステムでも、T-Kernelと互換性のあるアプリケーションを実装することができます。
- 3.MMU無しのCPUで動作させることを想定しています。

サポートCPU一覧

Software Package(一括ダウンロード)　µT-Kernel旧バージョン　µT-Kernel Release Note

USBSTICK 評価ボード



FM3 USBSTICK　検 索

詳細はwebで!!

仕様

製品名	SK-FM3-48PMC-USBSTICK (オーダー型格:MB2051-206-E)
搭載マイコン	MB9AF312K(ARM Cortex-M3)
最大周波	40MHz
FLASH ROM	128KB + 32KB
RAM	16KB
動作電圧	5.0V(USB給電)
USBインターフェース	USB2.0 Host/Function
実装デバイス	USBコネクタ 照度センサ LED 押しボタン
JTAGデバッグ	JTAG対応チップ搭載 IAR Embedded Workbench対応
製品構成	SK-FM3-48PMC-USBSTICK評価ボード セットアップCD USBケーブル

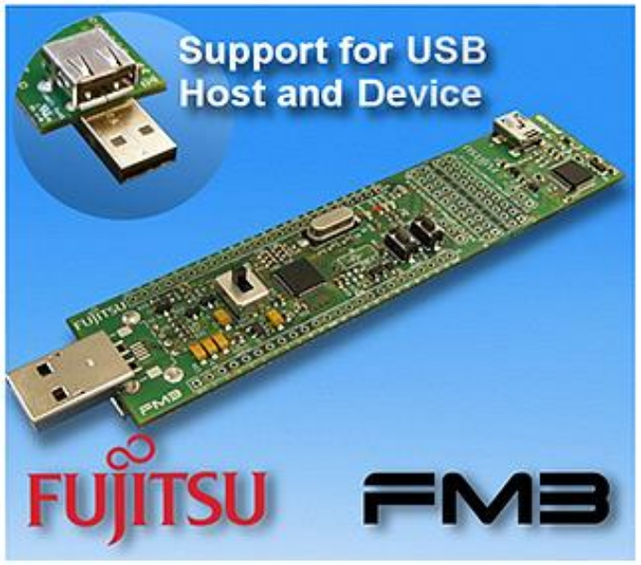
http://www.youtube.com/watch?v=U4qzNoDE6KY&feature=share

http://www.spansion.com/JP/Products/microcontrollers/32-bit-ARM-Core/pages/fm3-sk-fm3.aspx

USBSTICK 評価ボード

32ビット ARMコア
32ビット オリジナルコア
16ビット オリジナルコア
8ビット オリジナルコア
アプリケーション
技術解説

Spansion FM3ファミリ用USB評価ボード
SK-FM3-48PMC-USBSTICK（オーダー型格: MB2051-206-E）



USB-STICK専用お問い合わせ

ご購入はこちら>>

chip 1 stop　PERSONAL MEDIA CORP.

SK-FM3-48PMC-USBSTICKは、ARM社製「Cortex-M3コア」を搭載したマイクロコントローラ「FM3ファミリ」の低価格な評価ボードです。このボードには、マイコンの他にオンボードICEも搭載されているので、パソコンとUSBケーブルで接続するだけで、直ぐにJTAGデバッグを開始することが可能です。

ボード上のCortex-M3マイコンMB9AF312Kは、USB Host/Functionコントローラを内蔵しています。また、ボード上にはUSBに対応した周辺回路の他に、照度センサやLED、押しボタンも実装されているので、USB以外にも様々なアプリケーションの開発を行う事が可能です。

http://www.spansion.com/JP/Products/microcontrollers/32-bit-ARM-Core/pages/fm3-sk-fm3.aspx

Cortex-M3用のμT-Kernelが動く超低価格ボード

パーソナルメディア株式会社
http://www.t-engine4u.com/products/fm3_usbstick.html

【レポート】

- FM3 USBスティックボードにおけるμT-Kernelの利用方法(2013年9月5日) NEW
  FM3 USBスティックボードで、μT-Kernelを利用するための情報を追加しました。

【記事掲載】

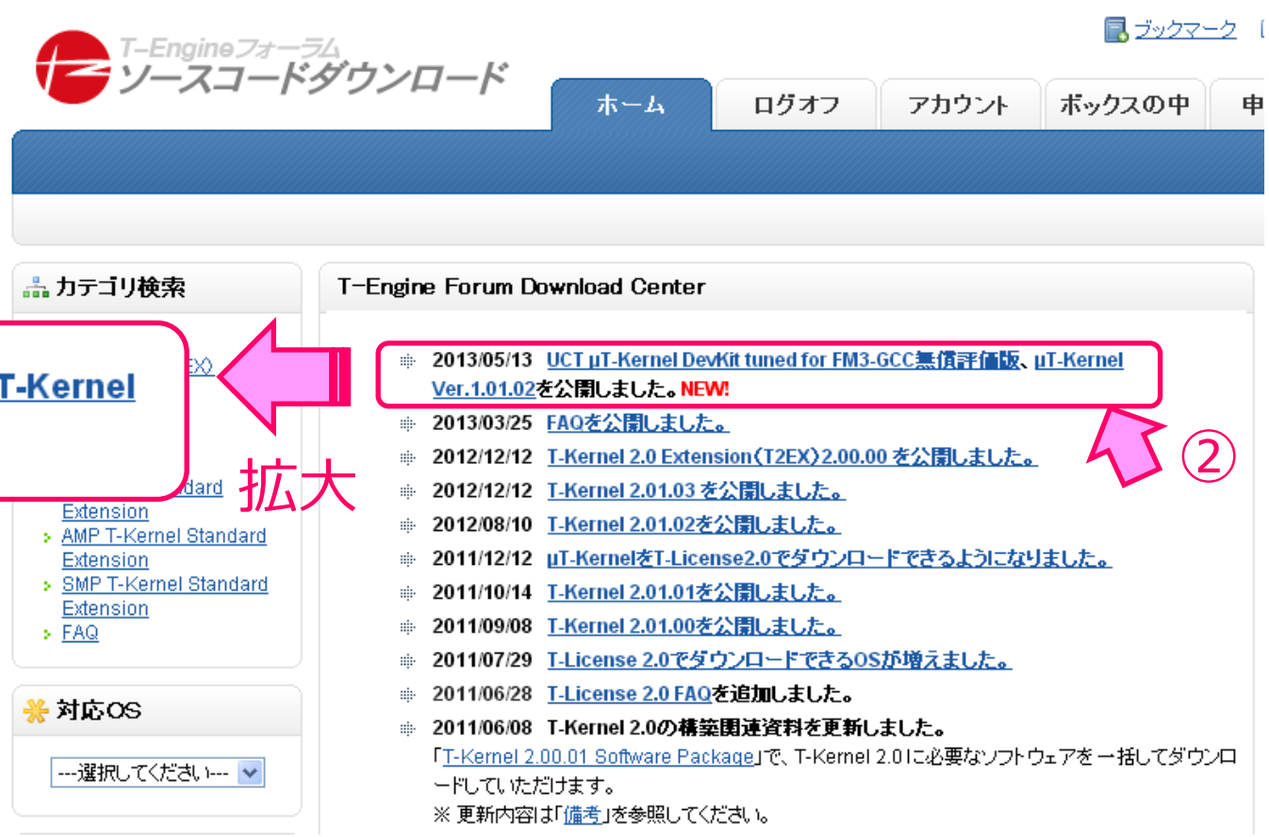
- 「TRONWARE(トロンウェア)」最新号 VOL.143に、『「FM3 USBスティックボード」でμT-Kernelを実行する』(p.36)が掲載されています。(2013年10月4日) NEW

FM3 USBスティックボードにおけるμT-Kernelの利用方法
http://www.t-engine4u.com/support/fm3_usbstick/install_utkernel.html

http://www.t-engine.org/ja/

①

http://www.t-engine.org/download/?language=jp



2013/05/13 UCT μT-Kernel DevKit tuned for FM3-GCC無償評価版、μT-Kernel Ver.1.01.02を公開しました。NEW!

拡大

②

http://www.t-engine.org/download/index.php?route=product/product&product_id=130

UCT μT-Kernel Software Package

画像を拡大できます

品番: UCT μTK Package

数量: 1 ボックスに入れる ③

④

ダウンロードボックス

1 x UCT μT-Kernel Software Package

ボックスへ 申請開始 ⑤

申請すると

ライセンス | 備考 | 関連ソフトウェア

UCT μT-Kernel DevKit tuned for FM3-GCC無償評価版利用条件

2013年5月13日制定即日施行
ユーシーテクノロジ株式会社

ダウンロードソフトウェア

オーダーID: サイズ: 251.68MB

名前: UCT_uTK1.01.02SoftwarePackage 残り: 10
申請日: 2013/05/26

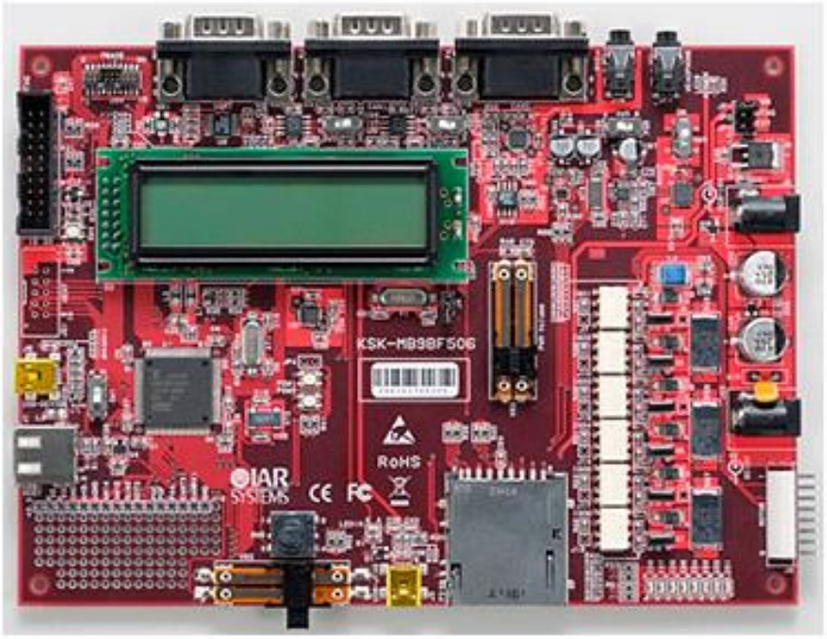
ダウンロード ⑥

ダウンロードできるようになる！！

今日実際にやってみる

FM3 スターターキット

搭載マイコン：MB9BF506R（Cortex-M3）



ボード部品	サンプルソフトウェア
LED	LED点滅
LCDパネル	LCDパネルに文字を表示
SDカード	（通信I/Fを利用したアクセス制御）
USB Host	USBマスストレージクラス
USB Function	USBマスストレージクラスでファイルをPCに表示
	USBコミュニケーションクラス 仮想COMポート
	USB HIDクラス（USBマウスデモ等）
CAN	CAN通信
ヘッドホンジャック	音声出力
モータドライブ回路	DCBLモータ制御（モータ別売り）

http://www.spansion.com/JP/Products/microcontrollers/32-bit-ARM-Core/pages/fm3-tool.aspx#starter

IAR SYSTEMS

【統合開発環境：EW-ARM】

高機能　利便性　信頼性　標準化

IAR Embedded Workbenchは、新しい製品の開発に役立つ、包括的で有効なツールです。

IAR Embedded Workbenchは、8ビット、16 ビット、32 ビットチップベースのアプリケーション開発用の高性能C/C++コンパイラ・デバッガツールスイートです。IARシステムズは、市場に出ている他のツールより多くのアーキテクチャのチップをサポートするため、世界中のリーディングチップベンダと連携しています。

IAR Embedded Workbenchは、現在から将来に向けてのすべての開発プロジェクトにとって、パワフルなツールです。

高機能　利便性　信頼性　標準化

IAR Embedded Workbenchには、1つの総合開発環境(IDE)にコンパイラ、アセンブラ、デバッガなどのコンポーネントがすべてシームレスに統合され、中断されないワークフローおよび単一のツールボックスを提供します。

IAR Embedded Workbenchは、高度でパワフルだけでなく、スマートな機能、ユーザフレンドリなインタフェースで使い勝手も考慮しています。IARシステムズは、チップベンダやRTOS/ミドルウェアベンダを含む広範囲なパートナー製品と連携、サポートを行っています。また、評価キットも提供しています。

IAR Embedded Workbenchは、効率の良い開発プロジェクトのためのユーザフレンドリなツールです。

高機能　利便性　信頼性　標準化

IAR Embedded Workbenchは、世界中のプログラマに出荷された100,000を超えるライセンスの実績を誇ります。それらの ユーザおよびパートナーからの情報および経験を、約30年に渡り製品にフィードバックしてきました。

組込み業界で有名な数多くの企業が、広範囲なテストが要求される医療機器を含む、安全性重視のアプリケーションで、IAR Embedded Workbenchを使用しています。さらに、高い信頼性は、独立系テストハウスや商用テストスイートにて検証されます。

ユーザは、www.iar.comの「マイページ」、ソフトウェアヘルプ機能、各国のセールスオフィスの技術サポートスタッフなどから、サポートが受けられます。保守契約を締結していただくことにより、ソフトウェアアップデートと技術サポートが保証されます。

IAR Embedded Workbenchは、開発プロジェクトの要求に応える高信頼性のツールです。

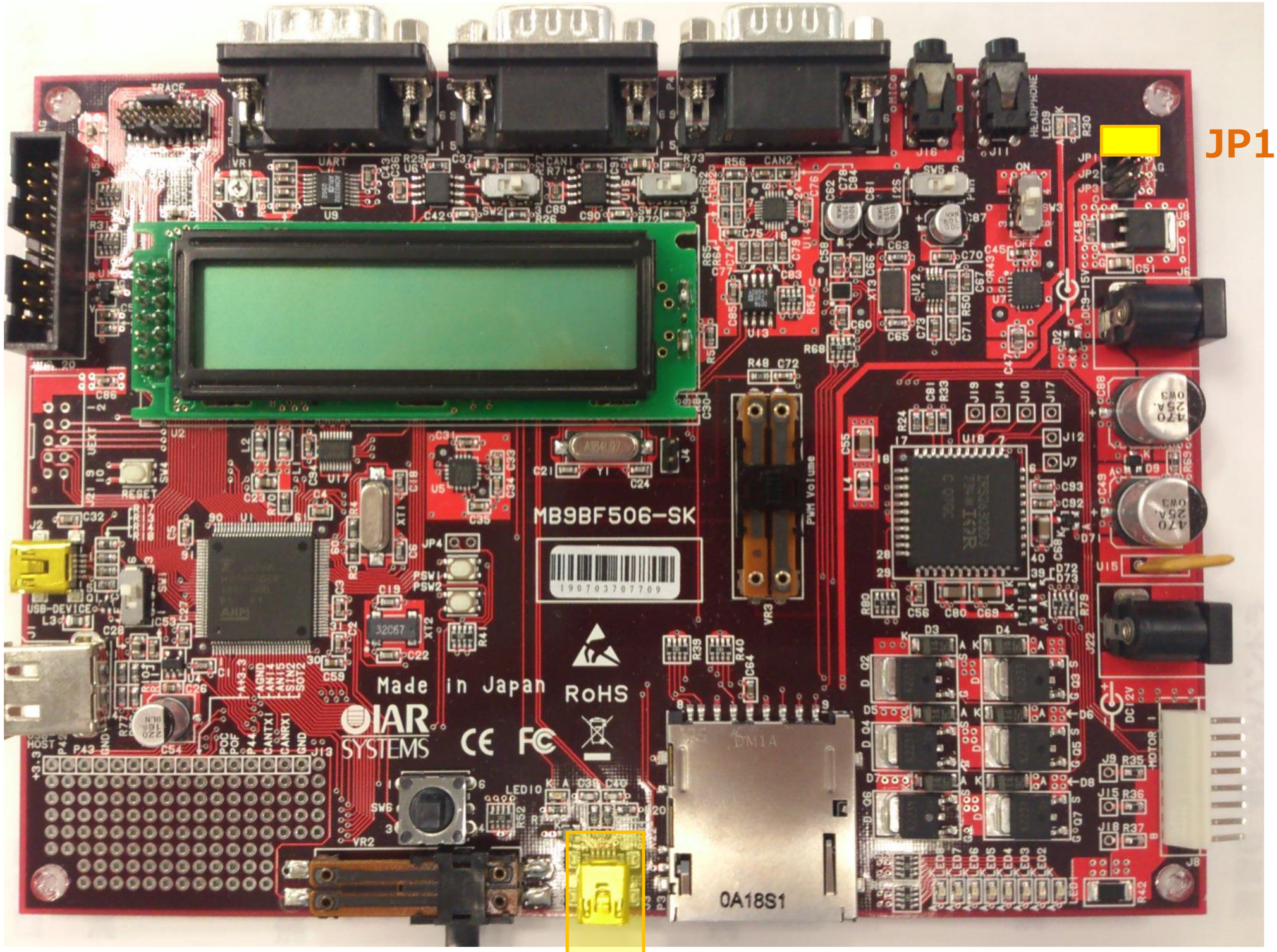
高機能　利便性　信頼性　標準化

複数のアーキテクチャーにおいて、個別のベンダーから分野・チーム・アプリケーション・製品・プロセッサをまたぐ広範囲な開発プロジェクトを管理していますか？製品開発を効率化し、現在および将来の製品に用いる半導体メーカを自由に選択したいと思いますか？

多くのグローバルな製造会社がIARシステムズのテクノロジーで組込みシステムの開発を標準化することを選択しています。IARシステムズのツールチェーンで標準化することにより、顧客は新製品を市場に出す効率・時間を大幅に改善することができます。全てのARMコアを含む全ての有名ベンダー、全ての関連アーキテクチャーにおいて、同一の環境で8ビット、16ビット、32ビットのマイコン間を自由に移行することができます。同一のツールチェーンで開発を標準化することで、ハードウェアや半導体メーカによらず、プロジェクトをまたいだコードの再利用が可能となり、トレーニング・メンテナンス・ライセンス管理の費用を削減することができます。

http://www.iar.com/jp/Products/IAR-Embedded-Workbench/

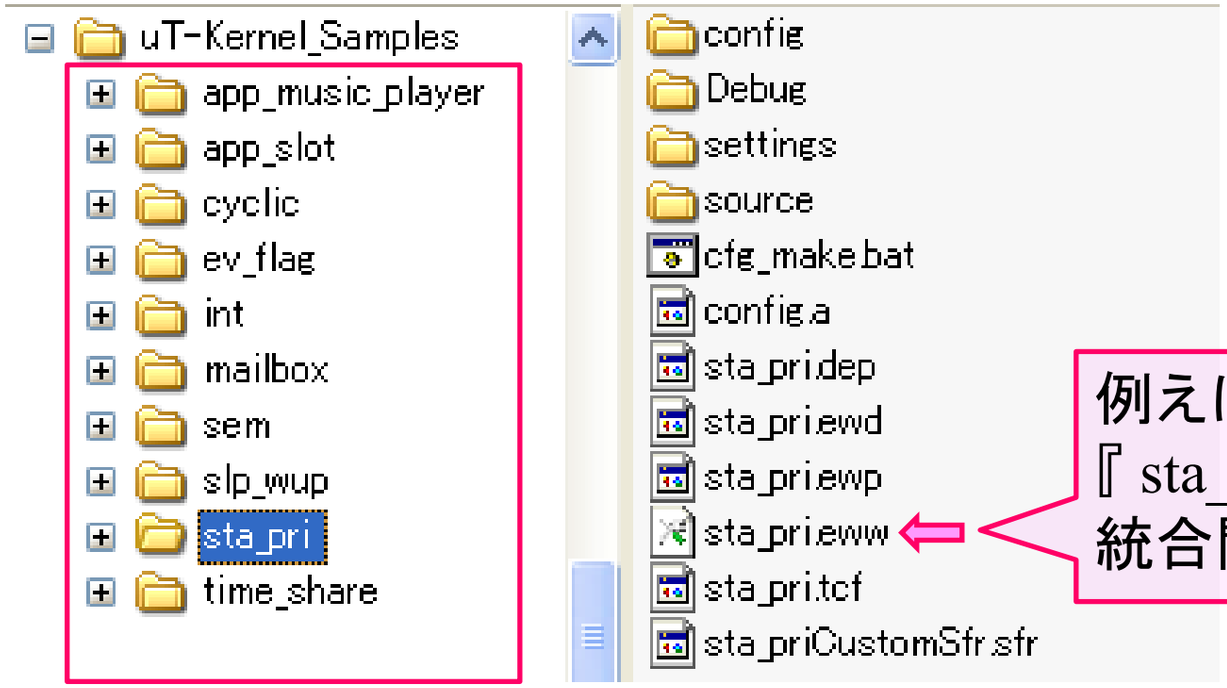
FM3 スターターキット準備



演習の進め方

使用するファイル

使用するデータ：C:¥ut-realos¥utkernel¥7-M¥uT-Kernel_Samples

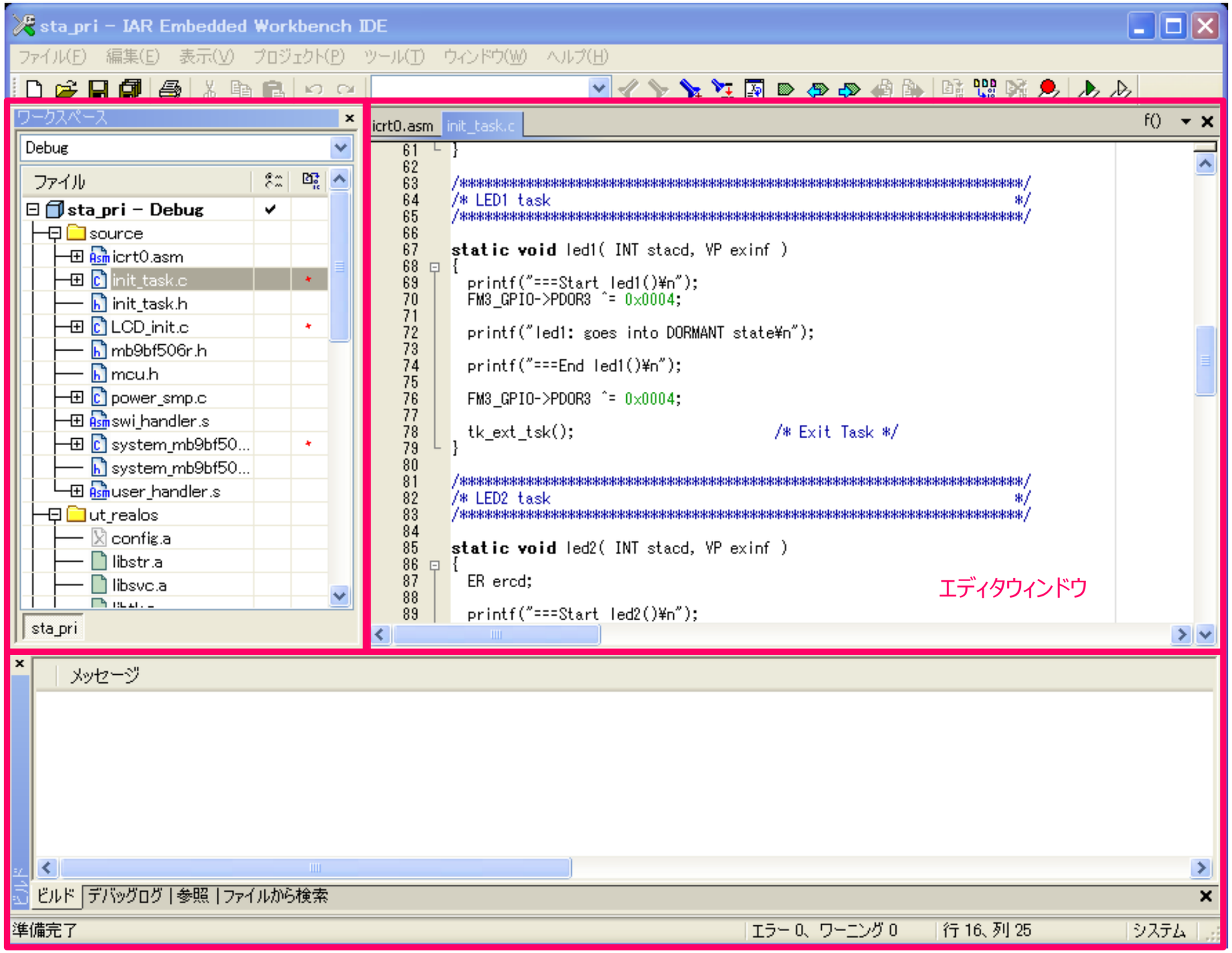


例えば，演習1『タスク起動と優先度』にとりかかる時，
『 sta_pri.eww 』をWクリックする.
統合開発環境 EW-ARM が起動される.

演習1-10

使用する統合開発環境

使用する統合開発環境：IAR Embedded Workbench for ARM （EW-ARM）



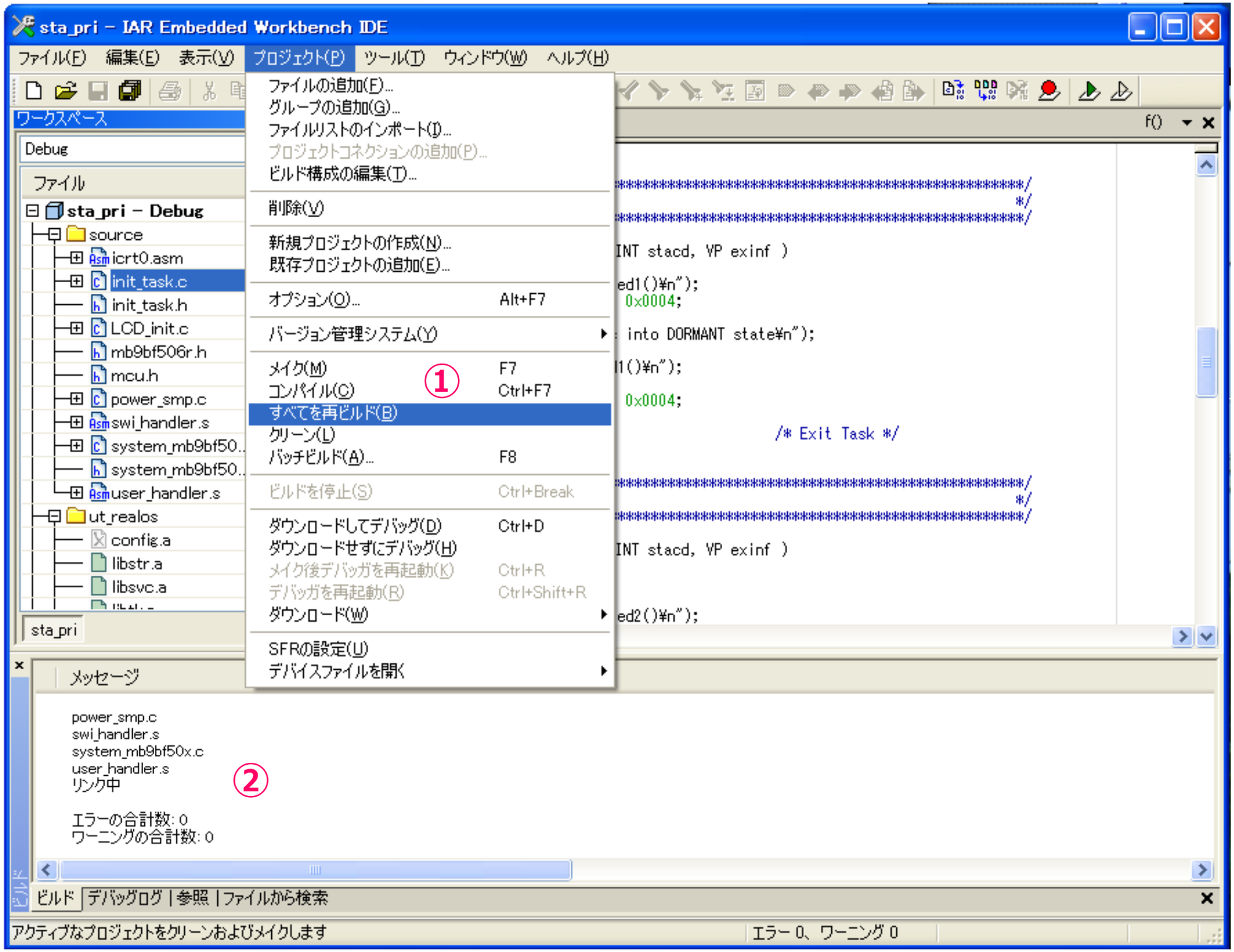
ワークスペース
ウィンドウ

エディタウィンドウ

メッセージ
ウィンドウ

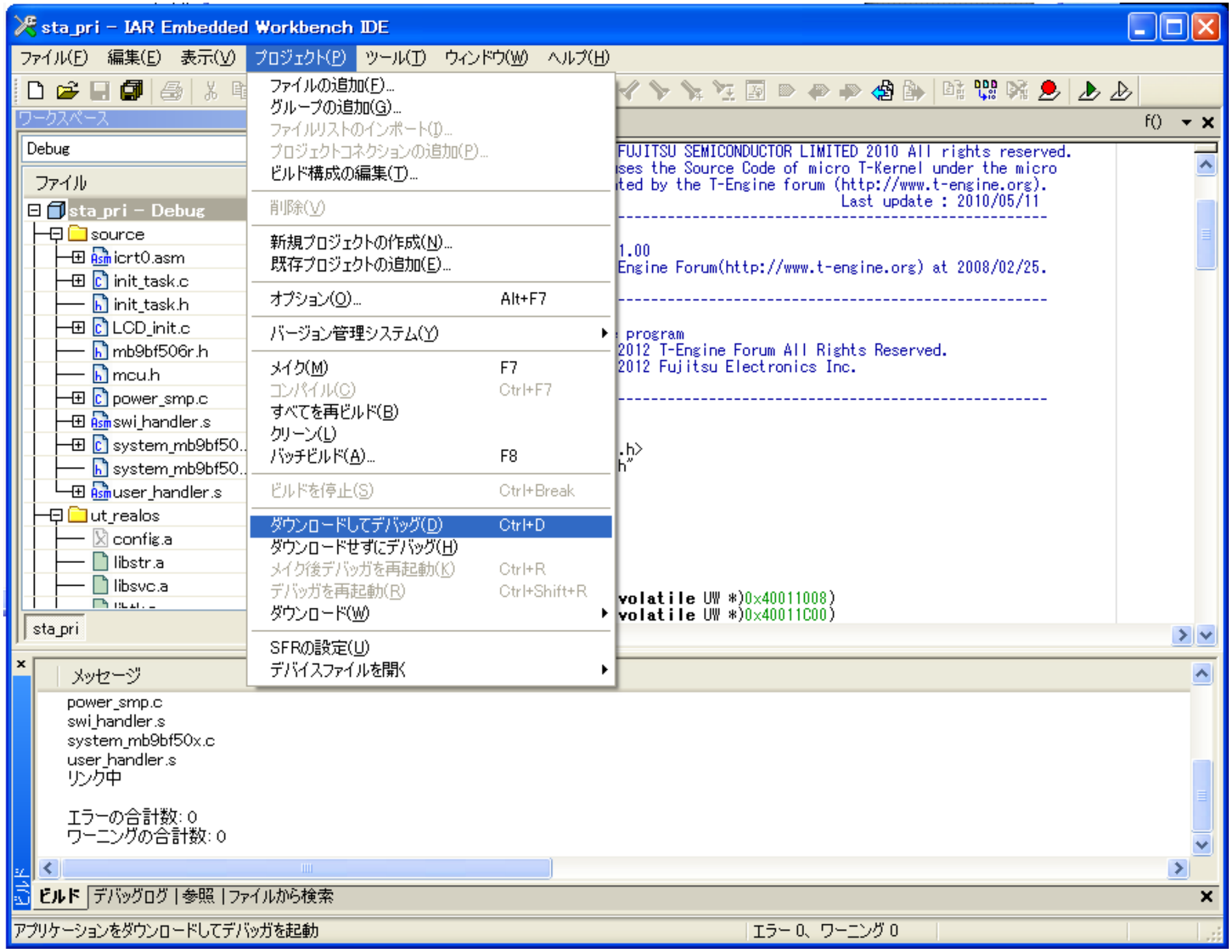
使用する統合開発環境

ビルド



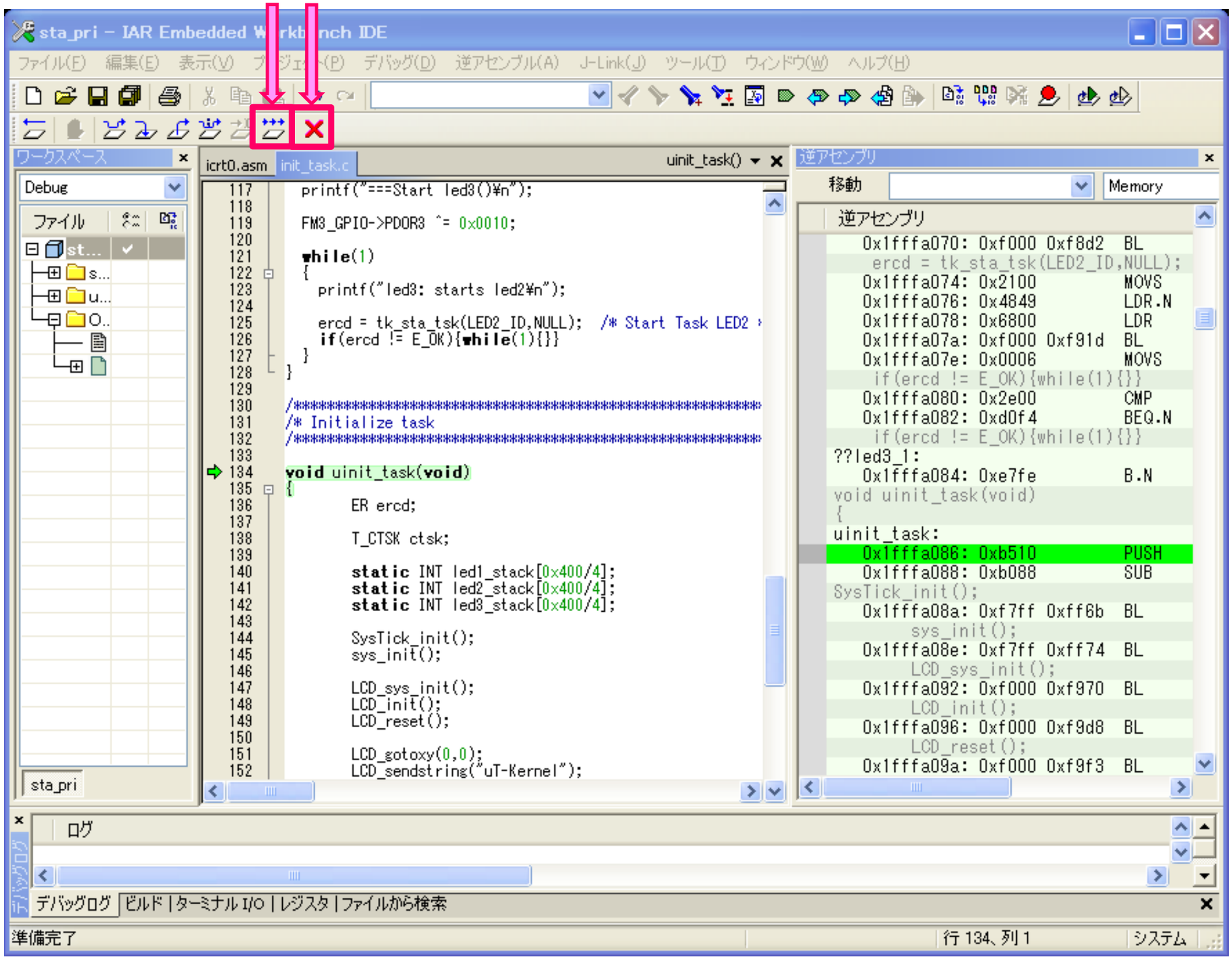
使用する統合開発環境

ダウンロードしてデバッグ



使用する統合開発環境

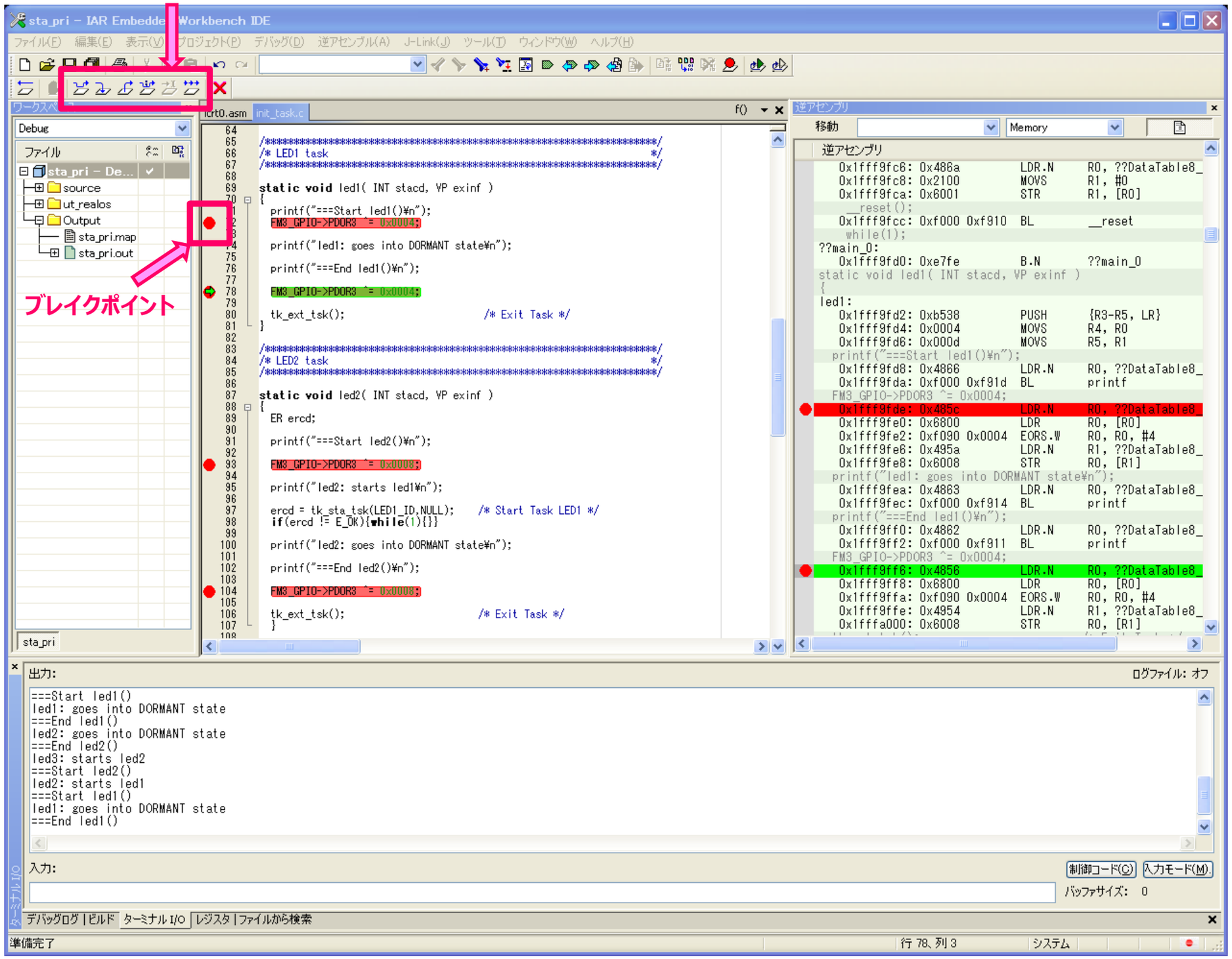
実行と停止



使用する統合開発環境

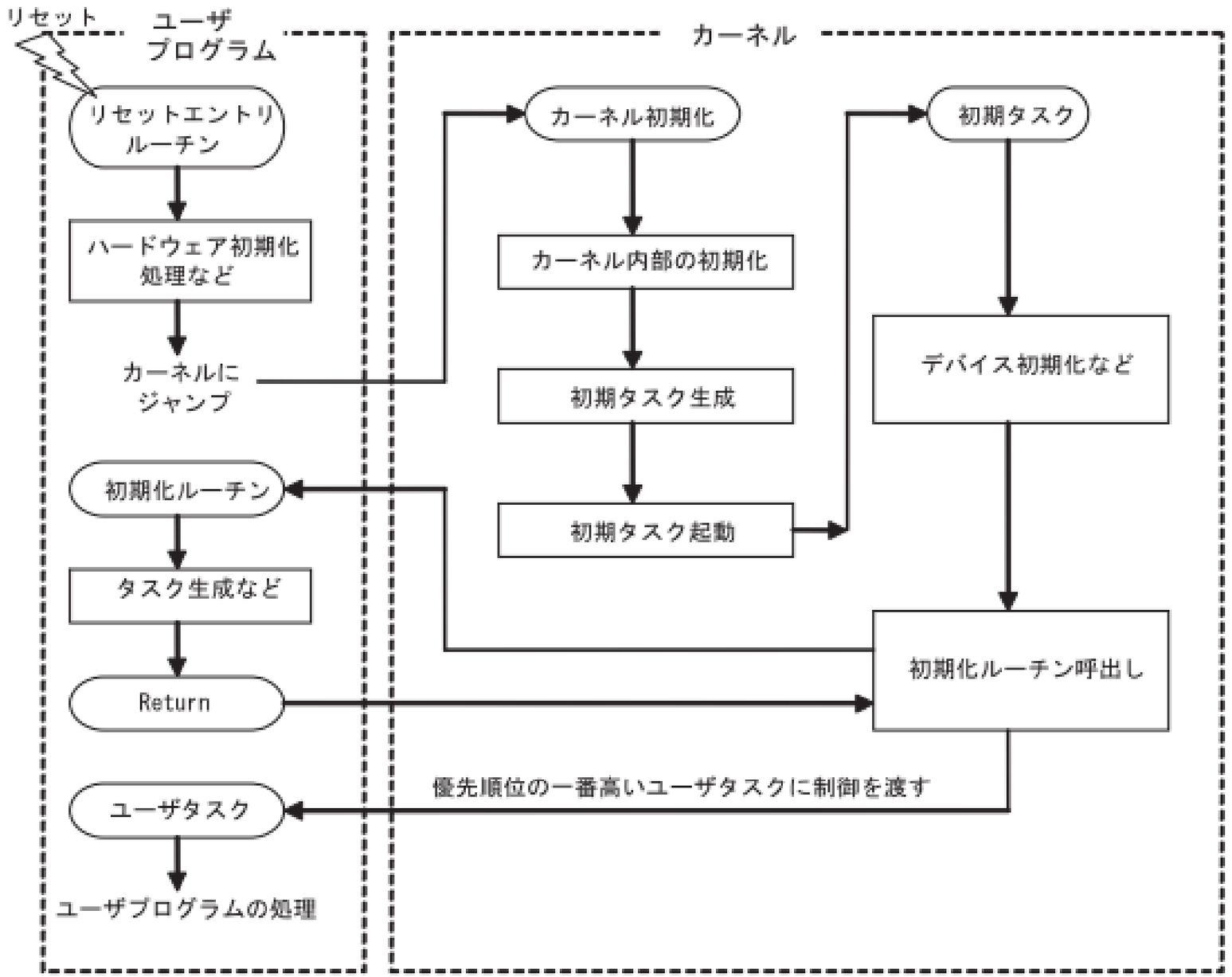
デバッグ

ステップ実行など



タスクスケジューリング（復習）

起動の流れ



図：起動の流れ

ハードウェアリセットが発生してから，ユーザプログラムのタスクに制御が移るまでの処理の流れ.
書記タスクでユーザプログラムの初期化ルーチンの呼出完了後，
ユーザプログラムの一番優先度の高いタスクに制御が渡る.

用語の解説

- タスク
  プログラムの並行実行の単位．並行して実行が行われるというのは，アプリケーションからみた概念的な動作であり，実装上はカーネルの制御のもとで，それぞれのタスクが時分割で実行される．

- カーネル
  各種システムコールの実処理や，スケジューラ，ディスパッチャなどOSの基本的な部分．

- スケジューリングとスケジューラ
  次に実行すべきタスクを決定する処理をスケジューリングと呼ぶ．
  スケジューリングを実現するカーネル内の機構をスケジューラと呼ぶ．

- ディスパッチとディスパッチャ
  プロセッサが実行するタスクを切り替えることをディスパッチと呼ぶ．
  ディスパッチを実現するカーネル内の機構をディスパッチャと呼ぶ．

- 優先順位
  優先度は，タスクやメッセージの処理順序を制御するために，アプリケーションによって与えられるパラメータである．優先順位は，処理の実行順序を明確にするために用いる概念である．
  タスク間の優先順位は，タスクの優先度に基づいて定められる．

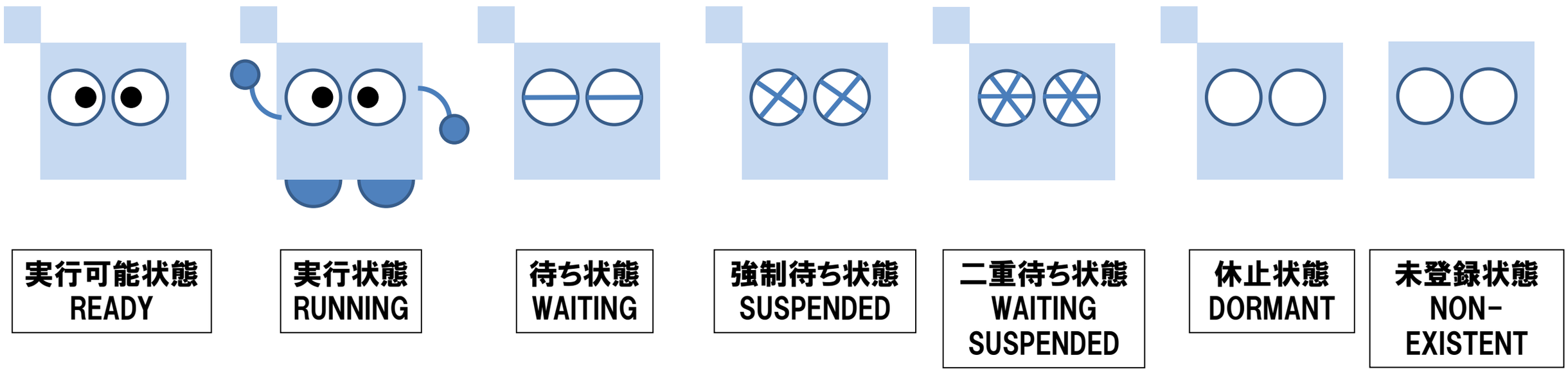
タスクの概要

◆タスクとは

- タスクは，並行処理するプログラムの単位
- 各タスクには優先度を指定

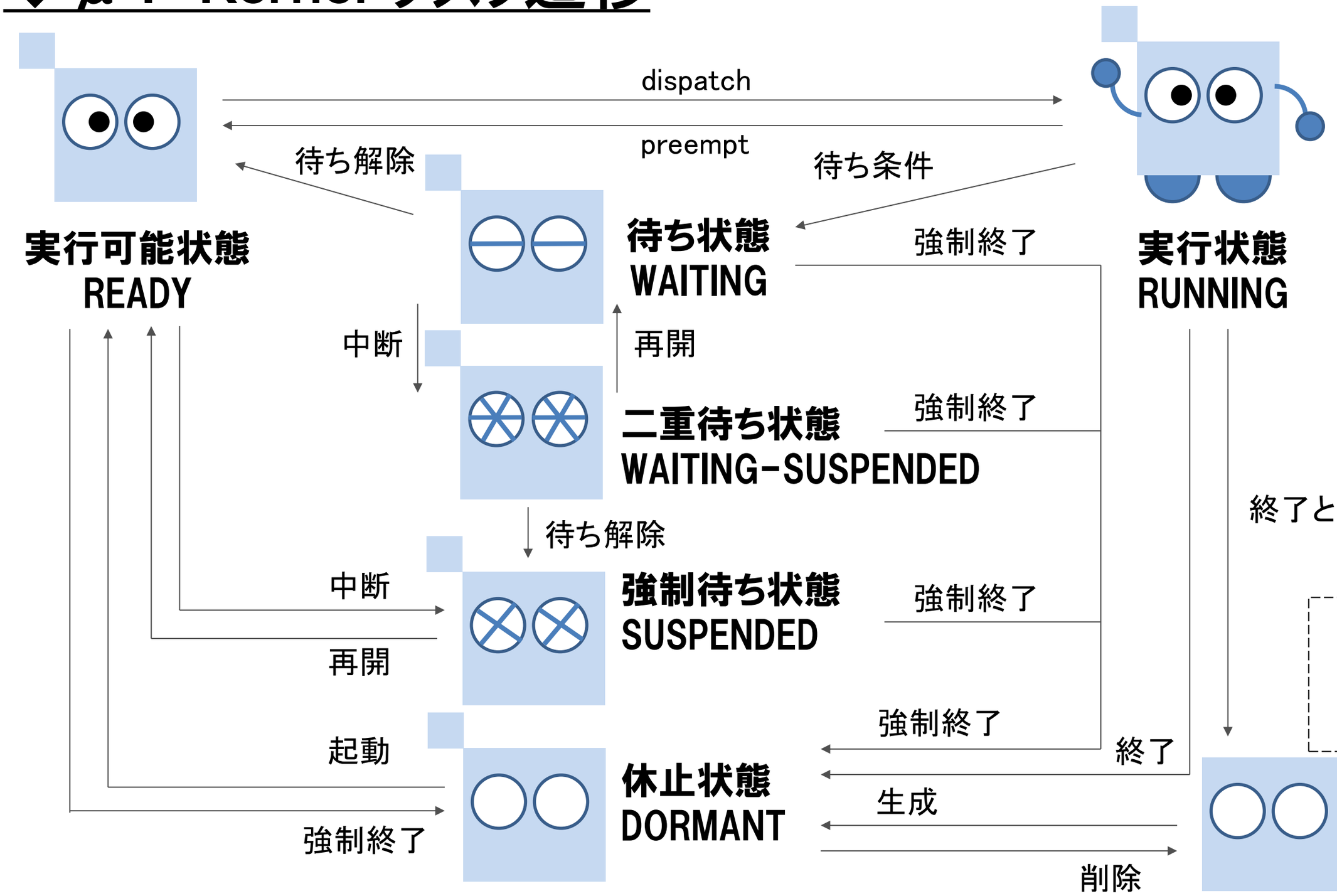
◆タスクの状態

- タスクの取りうる状態として次の7種の状態を定義している.

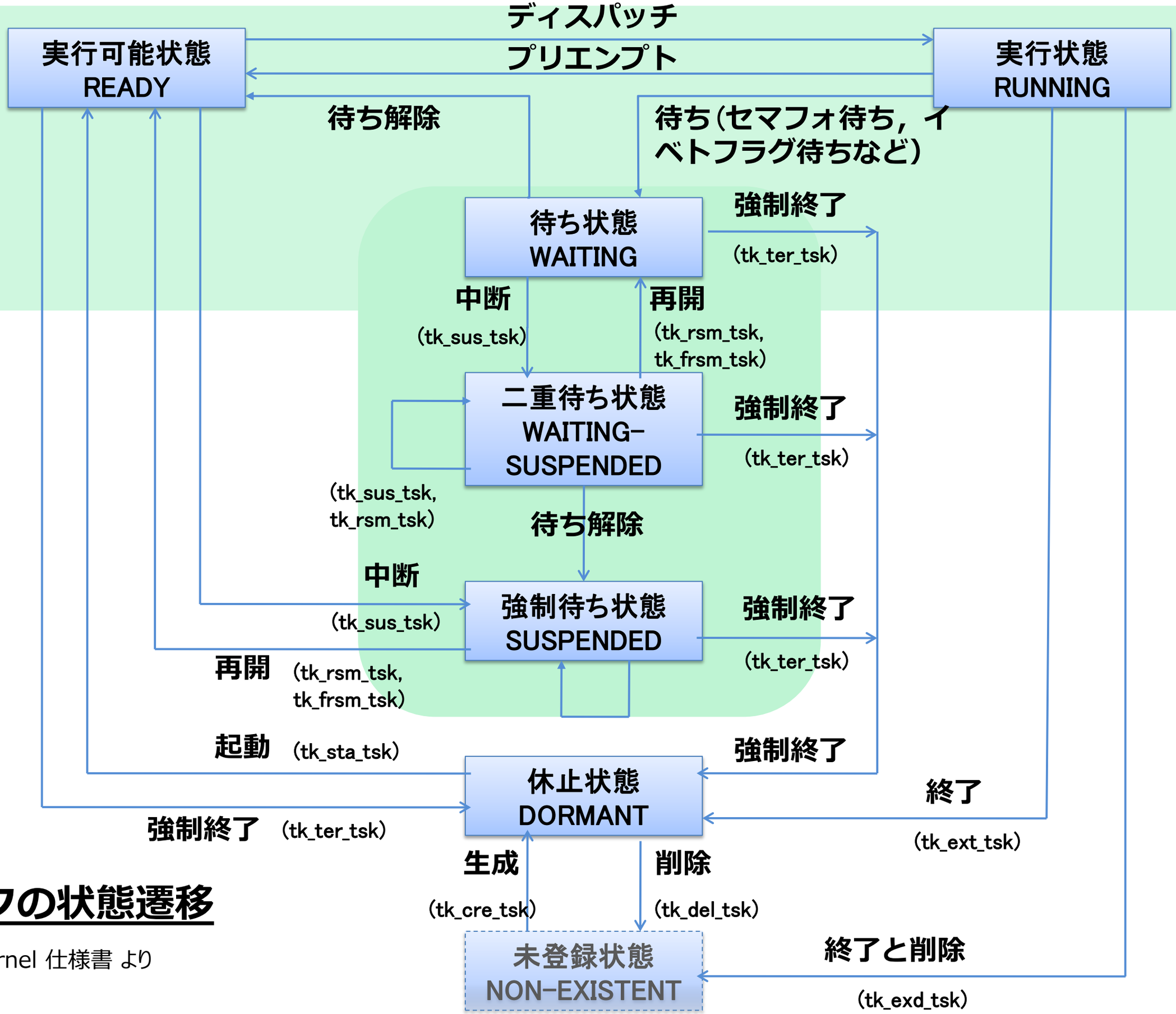


タスクの遷移状態

◆ μT-Kernel タスク遷移



タスクの状態遷移



μT-Kernel 仕様書 より

タスクのスケジューリング規則

μT-Kernelでは，タスク優先度に基づく
「優先度ベース スケジューリング方式」 を採用している.

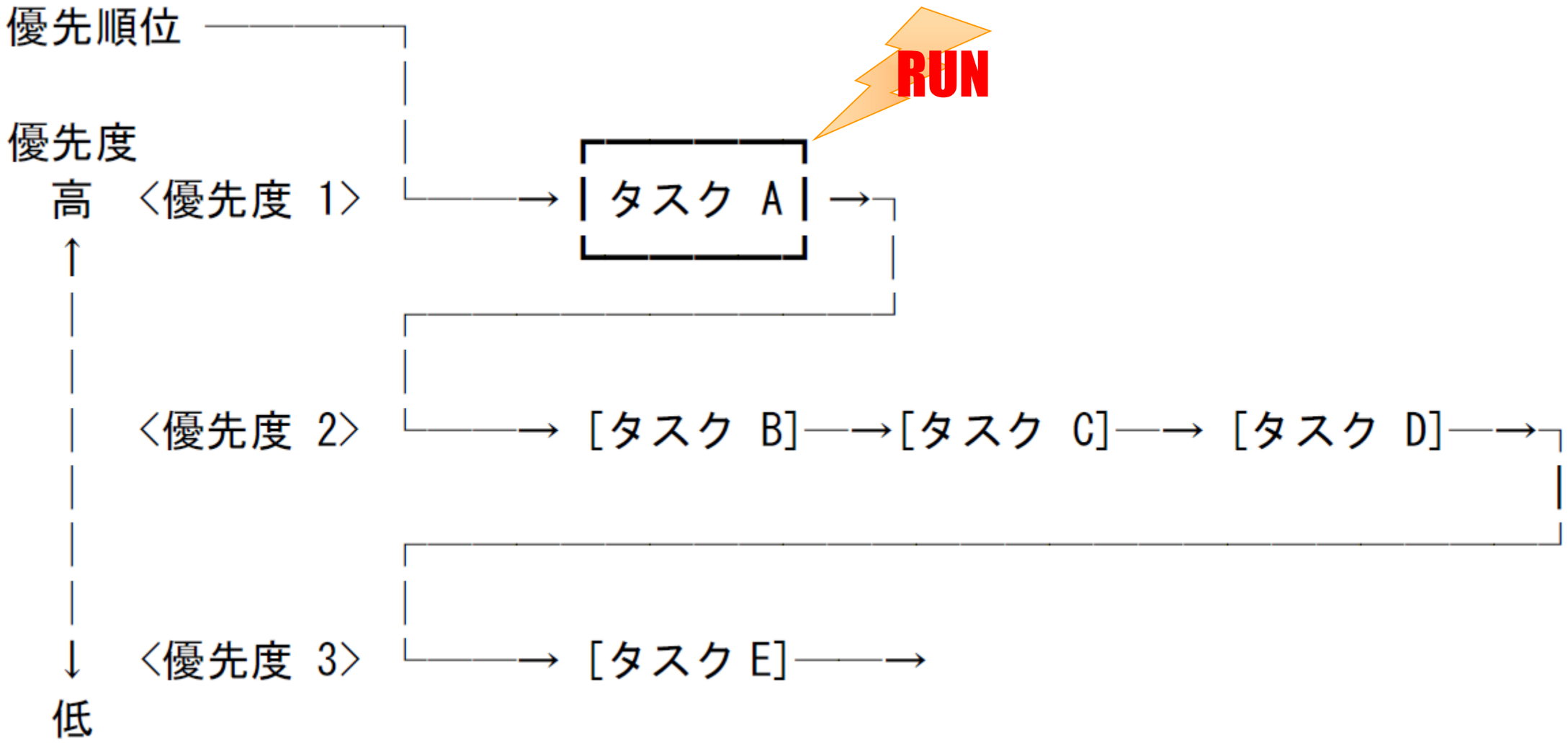
優先度ベーススケジューリング

- カーネルは，実行可能状態（READY）の中で 最も高い優先度 を持つタスクを実行させる.
- 同じ優先度のタスクが複数存在する場合，FCFS (First Come First Served) 方式（早い者勝ち！）で先に実行可能状態になったタスクが先に実行される.
- 優先度は 小さな値ほど高い.
- 優先度は，tk_chg_pri システムコールで変更可能.

タスクのスケジューリング規則

スケジューリングの例

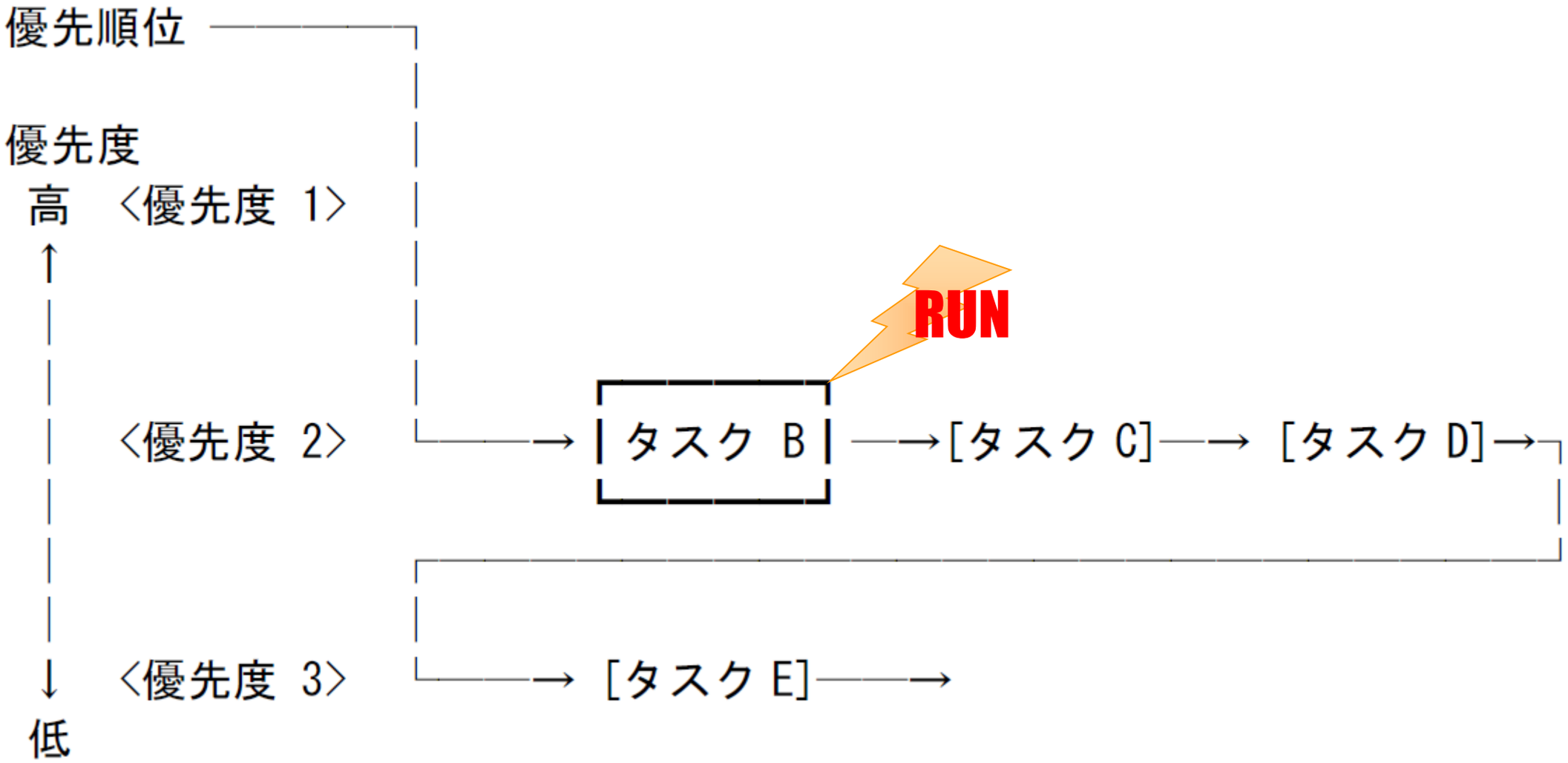
初期状態の優先順位



タスクのスケジューリング規則

スケジューリングの例

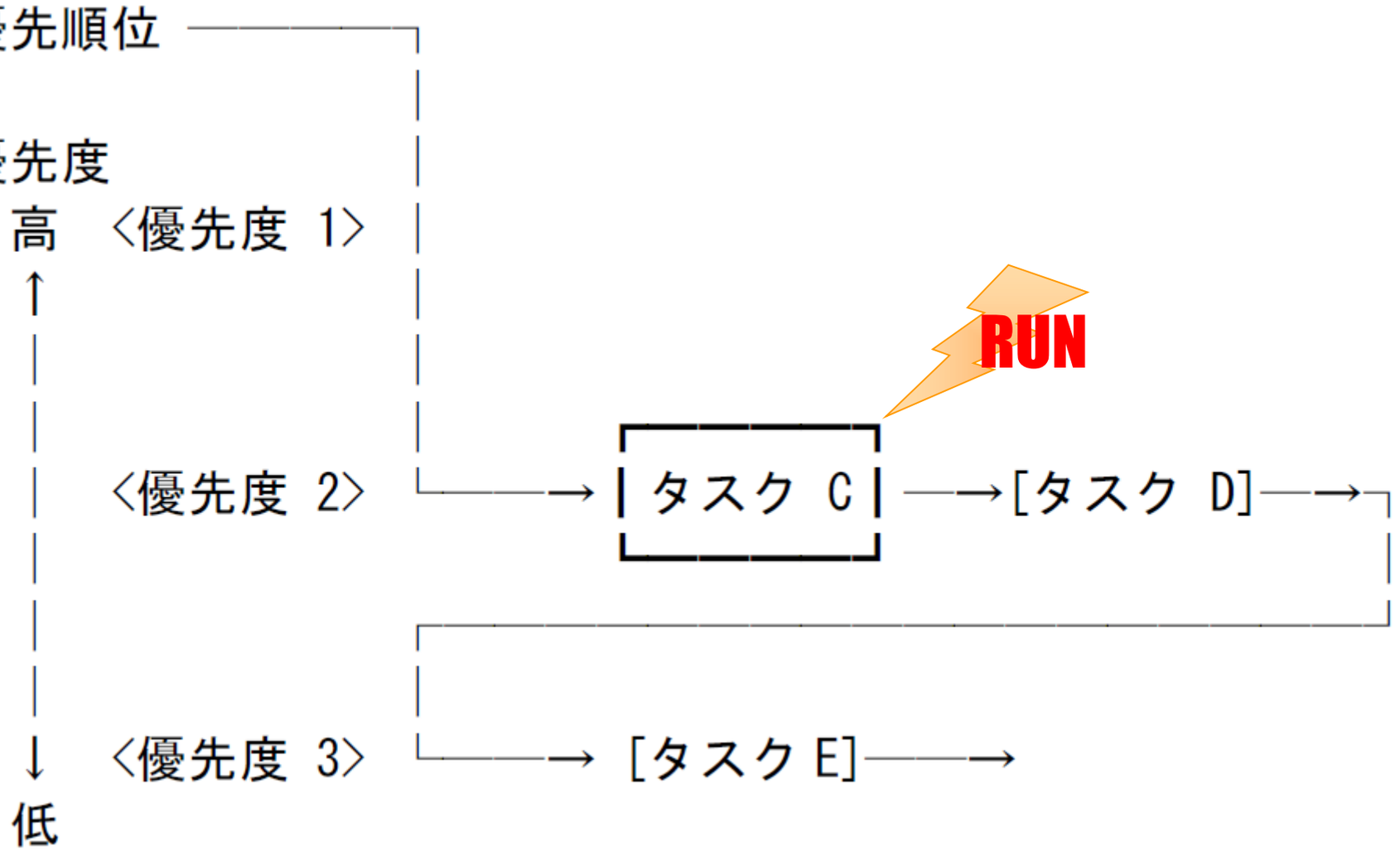
「タスクA」が「待ち状態」になった



タスクのスケジューリング規則

スケジューリングの例

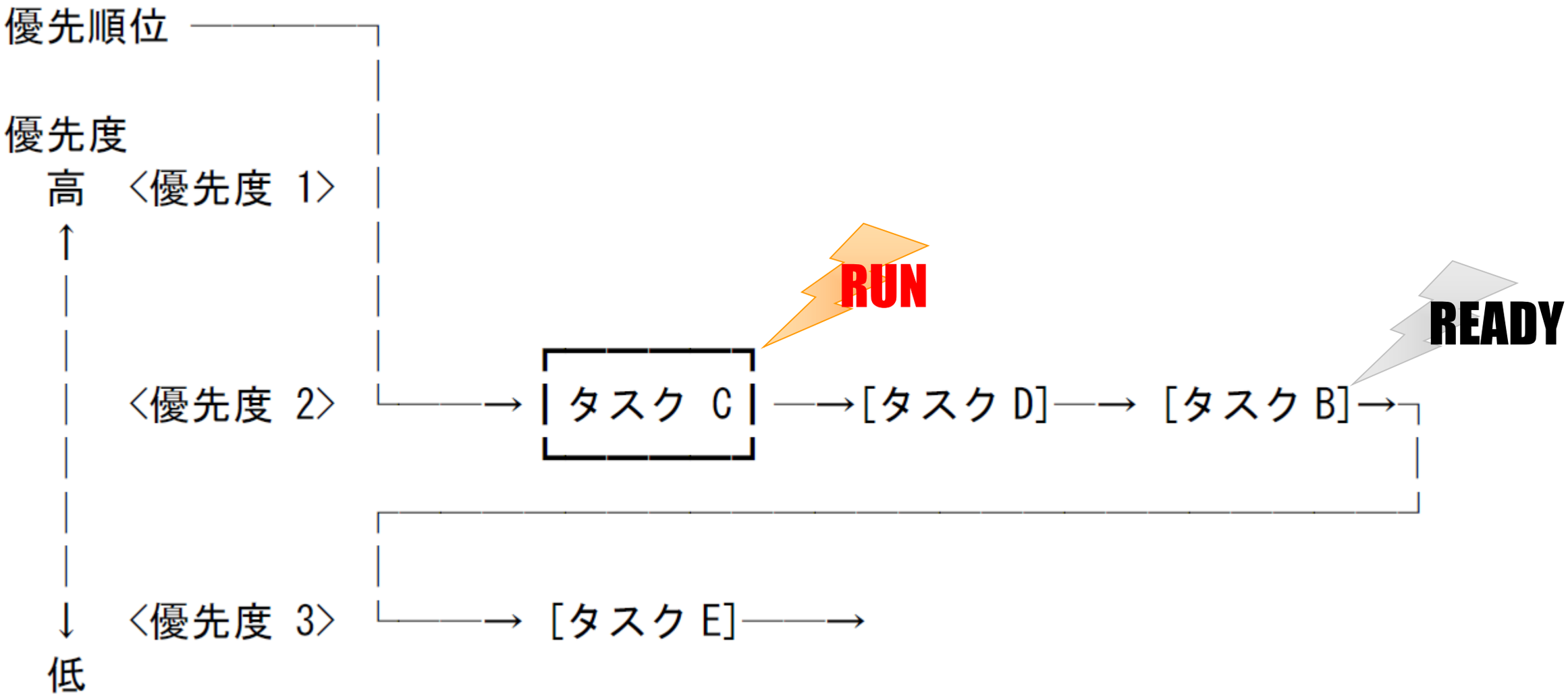
「タスクB」が「待ち状態」になった



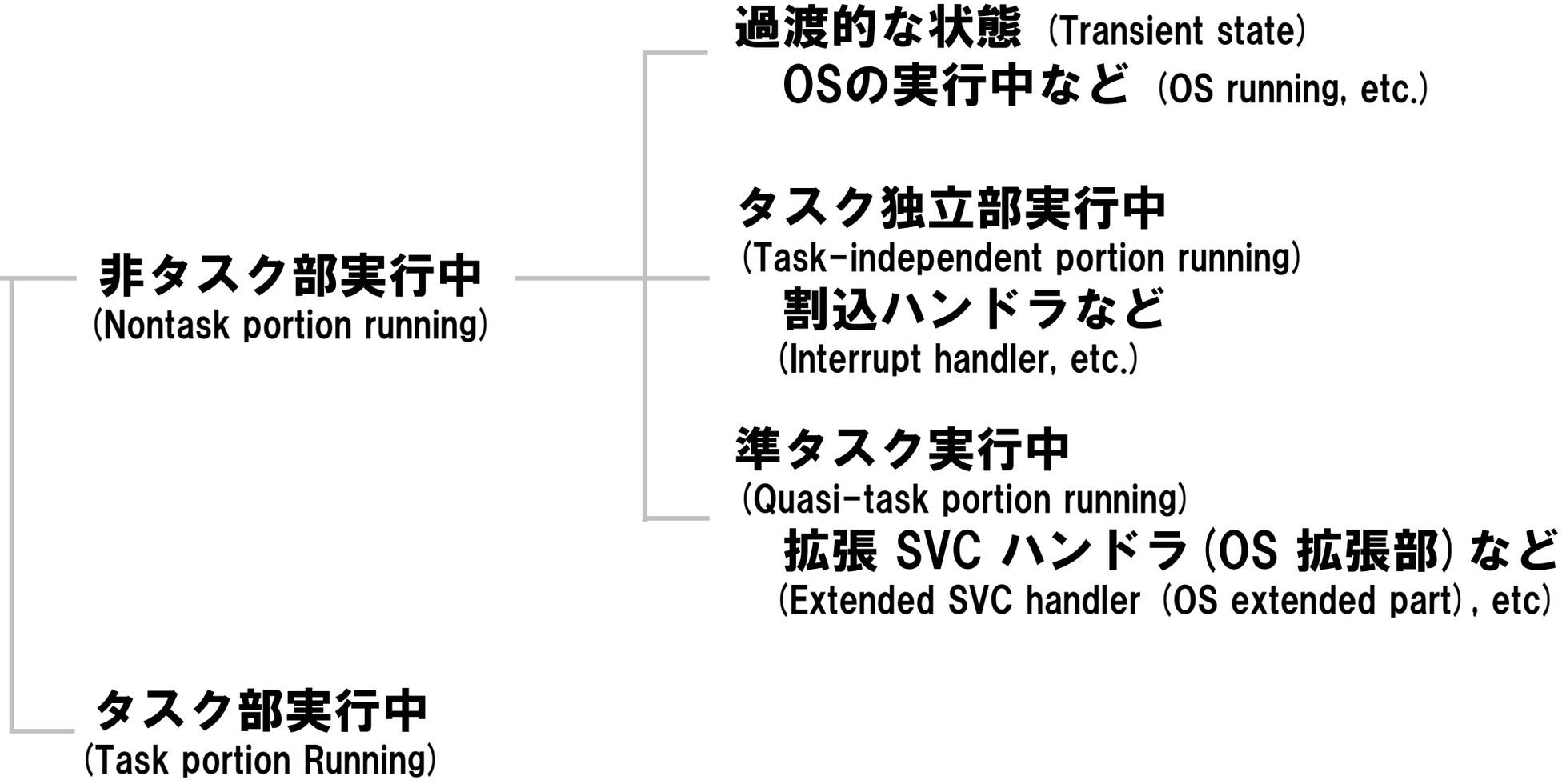
タスクのスケジューリング規則

スケジューリングの例

「タスクB」が「待ち解除」された



システム状態



第3章
μT-Kernel の機能

µT-Kernel の機能一覧

- タスク管理機能
- タスク付属同期機能
- 同期・通信機能
  - セマフォ
  - イベントフラグ
  - メールボックス
- 拡張同期・通信機能
  - ミューテックス
  - メッセージバッファ
  - ランデブポート
- メモリプール管理機能
  - 固定長メモリプール
  - 可変長メモリプール
- 割込み管理機能
  - 割込みハンドラ機能
  - CPU割込み制御
- 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
- システム状態管理機能
- サブシステム管理機能
- デバイス管理機能
- デバッグサポート機能

1. タスク管理機能

タスク管理機能とは？

- タスク管理機能は，タスク状態を直接的に操作/参照するための機能.

システムコール

タスク生成 *Create Task*

```
ID tskid = tk_cre_tsk (T_CTSK *pk_ctsk);
```

タスク削除 *Delete Task*

```
ER ercd = tk_del_tsk (ID tskid);
```

タスク起動 *Start Task*

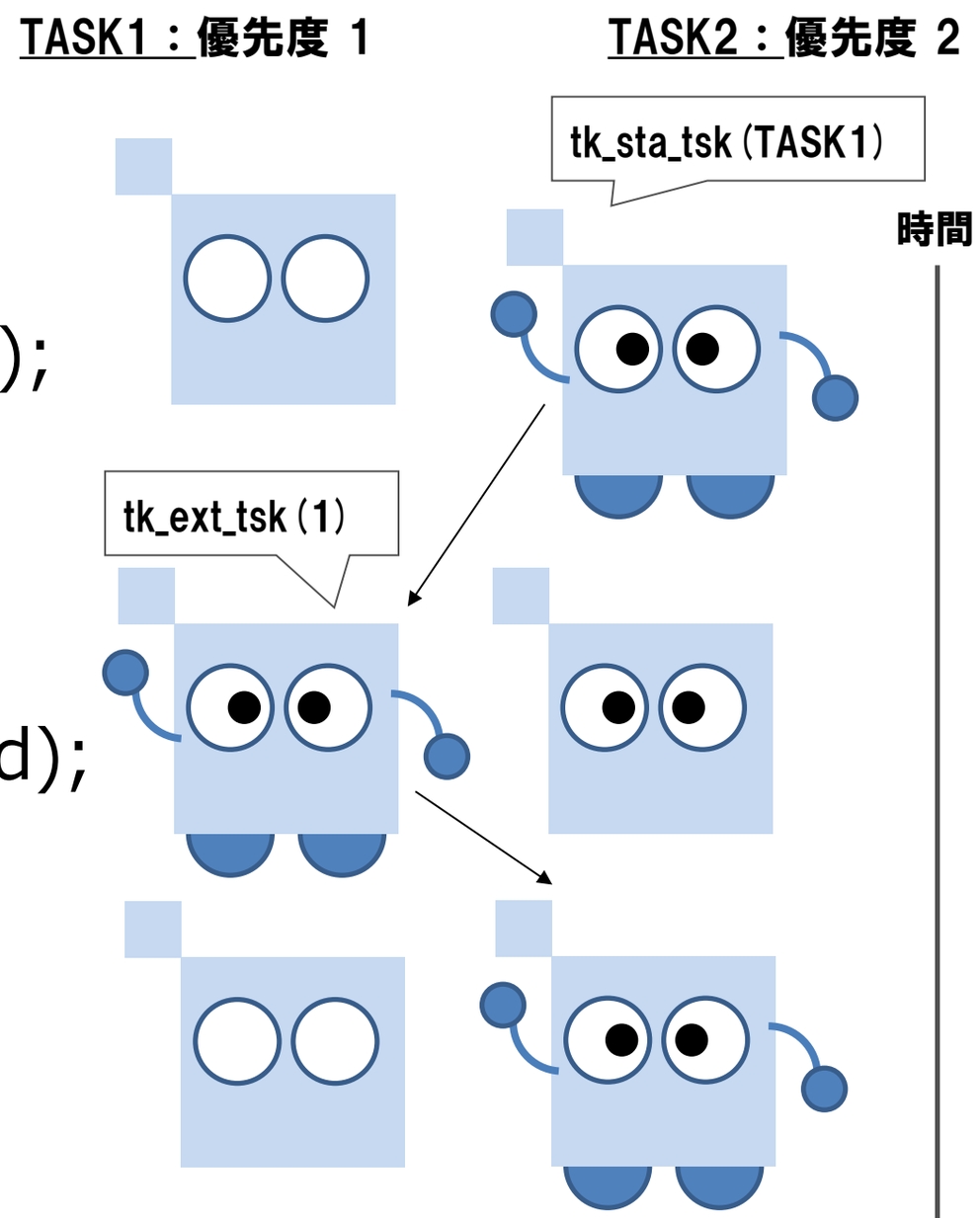
```
ER ercd = tk_sta_tsk (ID tskid, INT stacd);
```

自タスク終了 *Exit Task*

```
void tk_ext_tsk (void);
```

自タスク終了と削除

```
void tk_exd_tsk (void);
```



1. タスク管理機能

システムコール

他タスクの強制終了 *Terminate Task*

```
ER ercd = tk_ter_tsk (ID tskid);
```

タスク優先度変更 *Change Task Priority*

```
ER ercd = tk_ c h g _pri (ID tskid, PRI tskpri);
```

タスクレジスタの取得 *Get Task Registers*

```
ER ercd = tk_get_reg (ID tskid, T_REGS *pk_regs, T_EIT *pk_eit,
T_CREGS *pk_cregs);
```

タスクレジスタの設定 *Set Task Registers*

```
ER ercd = tk_set_reg (ID tskid, T_REGS *pk_regs, T_EIT *pk_eit,
T_CREGS *pk_cregs);
```

タスク状態参照 *Reference Task Status*

```
ER ercd = tk_ref_tsk (ID tskid, T_RSTK *pk_rtsk);
```

1. タスク管理機能

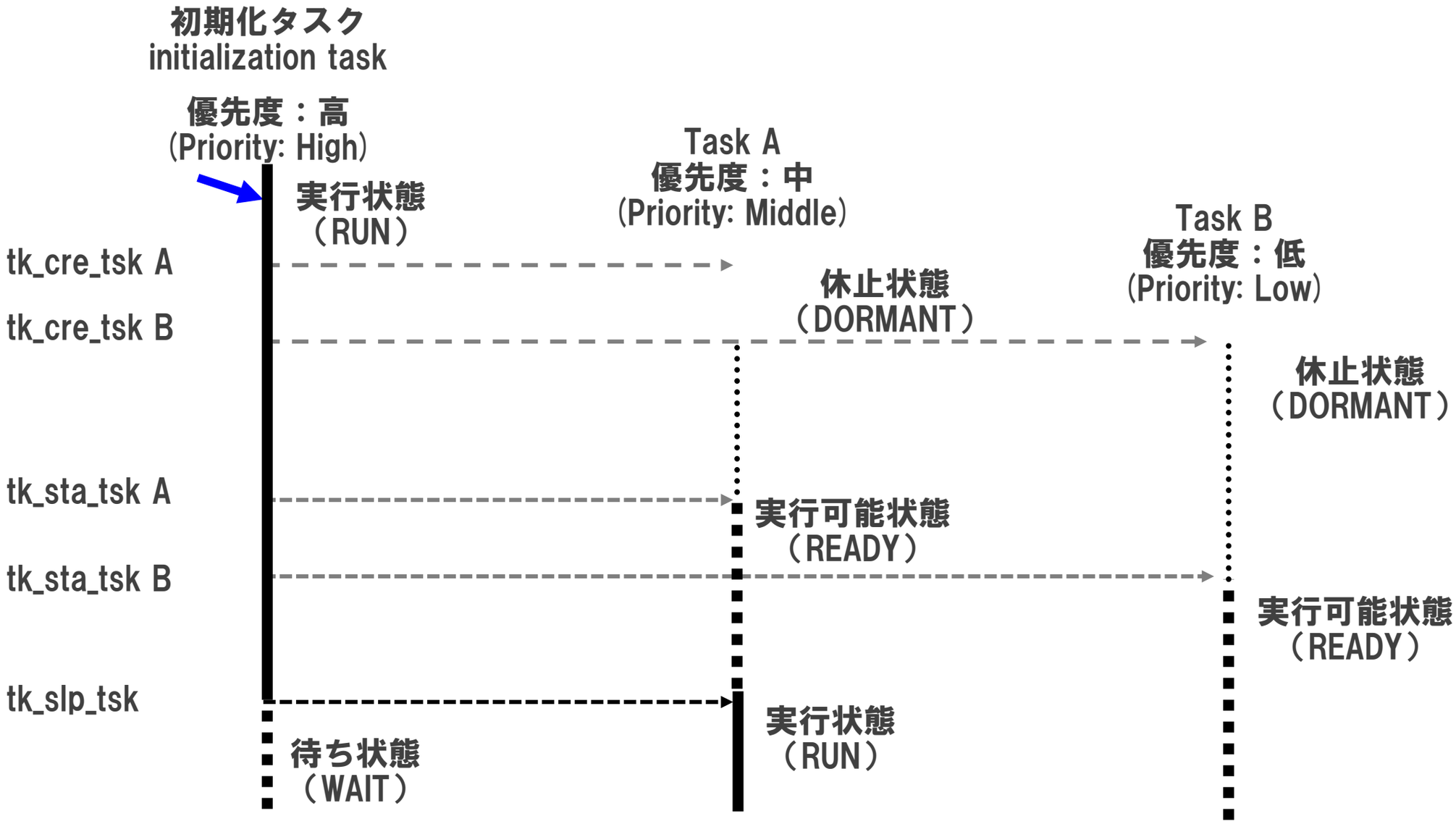
初期化タスクの処理例

```
{
  ER       ercd;
  T_CTSK   ctsk_A, ctsk_B;              /* タスク生成情報 */
  ID       tskid_A, tskid_B;            /* タスクID       */
     ここでタスク構造体 ctsk_A, ctsk_B を設定
  tskid_A = tk_cre_tsk(&ctsk_A);        /* タスク生成     */
  tskid_B = tk_cre_tsk(&ctsk_B);
  ercd  = tk_sta_tsk(tskid_A, mbxId);   /* タスク起動     */
  ercd  = tk_sta_tsk(tskid_B, mbxId);
  ercd  = tk_slp_tsk(TMO_FEVR);         /* タスクスリープ */
}
```

※エラー処理は入っていません.

1. タスク管理機能

初期化タスクの流れ



2. タスク付属同期機能

タスク付属同期機能とは？

- タスク付属同期機能は，タスクの状態を直接的に操作することによって同期を行うための機能.

システムコール

自タスクを起床待ち状態へ移行 *Sleep Task*

```
ER ercd = tk_slp_tsk (TMO tmout);
```

他タスクの起床 *Wakeup Task*

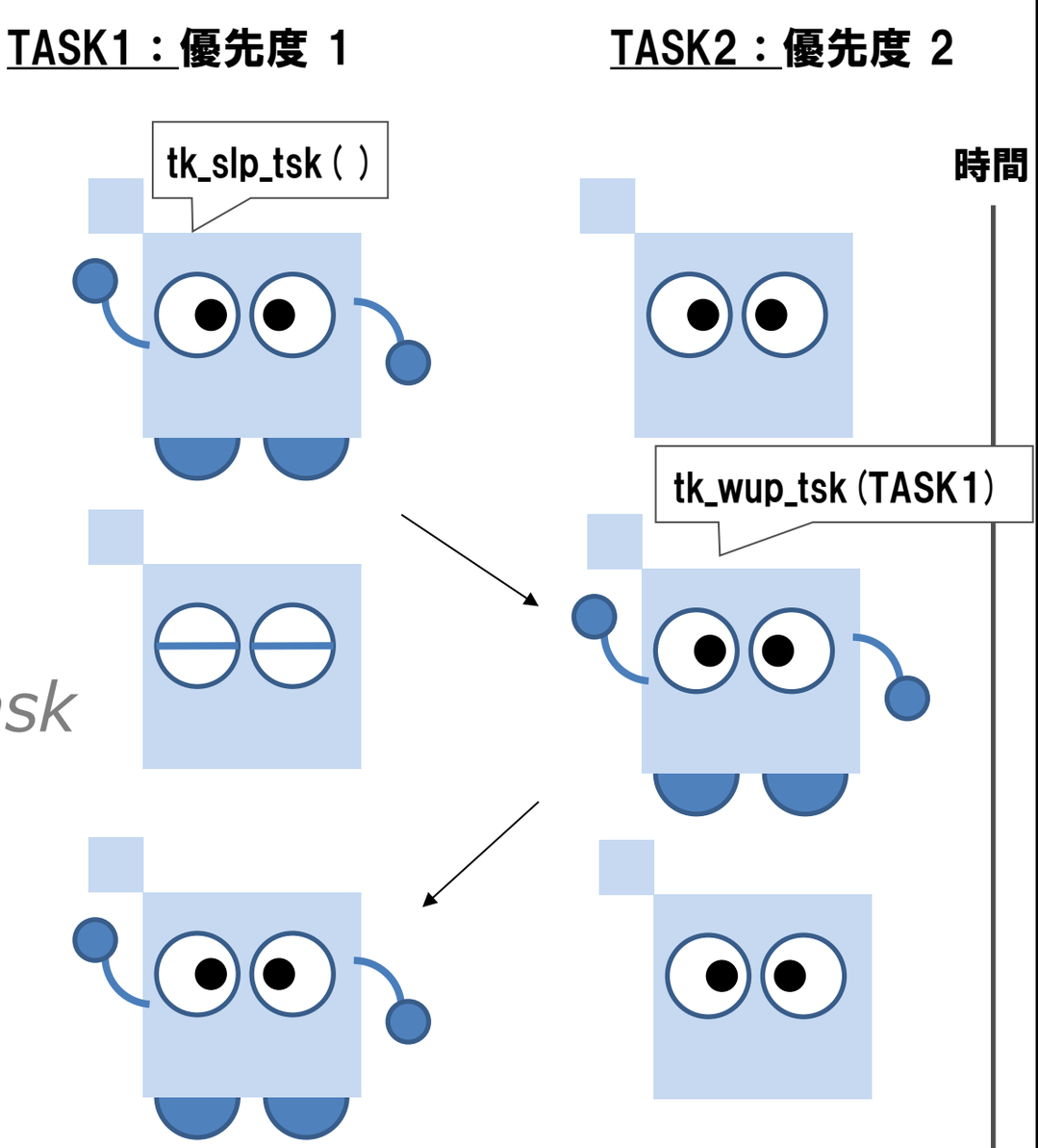
```
ER ercd = tk_wup_tsk (ID tskid);
```

タスクの起床要求を無効化 *Cancel Wakeup Task*

```
INT wupcnt = tk_can_wup (ID tskid);
```

他タスクの待ち状態解除 *Release Wait*

```
ER ercd = tk_rel_wai (ID tskid);
```



【 演習1 タスク起動と優先度 】

■ プロジェクトフォルダ
C:¥ut_realos¥utkernel¥7-M¥uT-Kernel_Samples¥sta_pri

■ プロゴラム概要
・タスクは，led1(優先度1)，led2(優先度2)，led3(優先度3) から構成される
・初期化タスク uinit_task はタスクを動的に生成し，led3 を開始する
・各タスクがより優先順位の高いタスクの起動と字タスクの終了を繰り返す

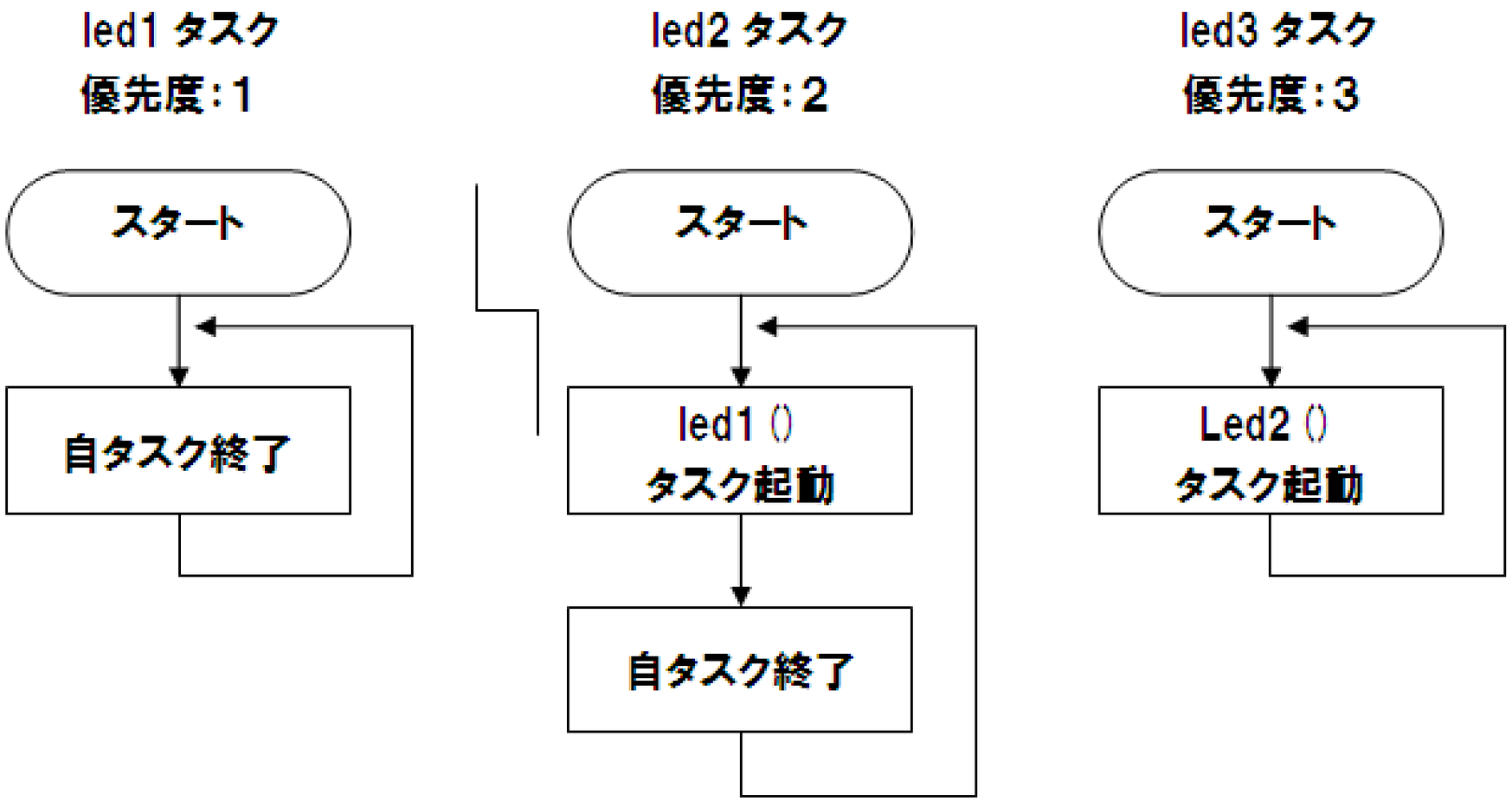


図1．各タスクのフローチャート

【 演習1 タスクチャートを書いてみよう 】

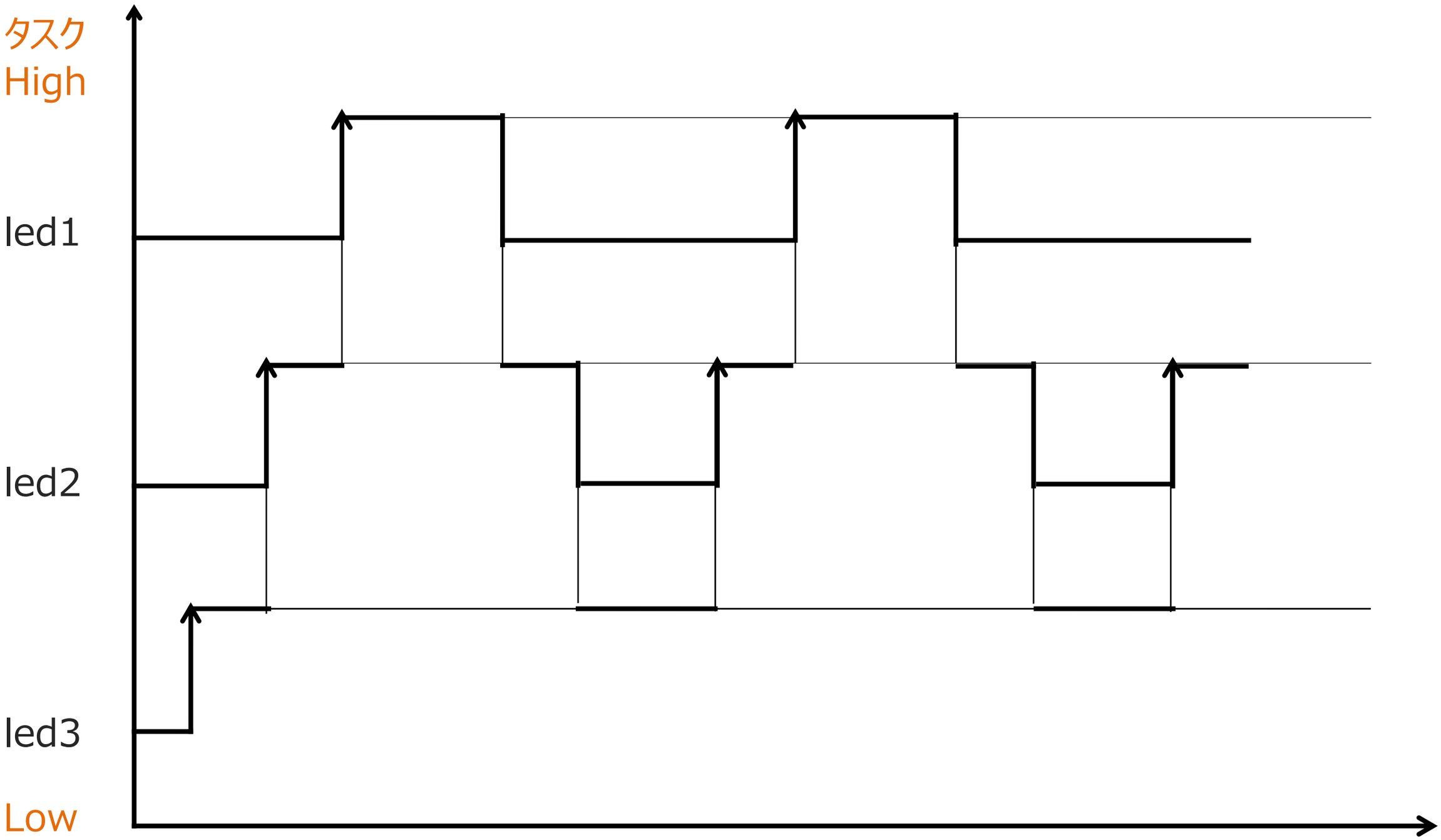


図2. タスクチャート

2. タスク付属同期機能

システムコール

他タスクを強制待ち状態へ移行 *Suspend Task*

```
ER ercd = tk_sus_tsk (ID tskid);
```

強制待ち状態のタスクを再開 *Resume Task*

```
ER ercd = tk_rsm_tsk (ID tskid);
```

強制待ち状態のタスクを強制再開 *Force Resume Task*

```
ER ercd = tk_frsm_tsk (ID tskid);
```

タスク遅延 *Delay Task*

```
ER ercd = tk_dly_tsk (RELTIM dlytim);
```

【 演習2 起床待ちと起床 】

■ プロジェクトフォルダ
C:¥ut_realos¥utkernel¥7-M¥uT-Kernel_Samples¥slp_wu

■ プロゴラム概要
・タスクは，led1(優先度6)，led2(優先度5)，led3(優先度4) から構成される
・初期化タスク uinit_task はタスクを動的に生成し，他のタスクに起床されるまで待つ．
他タスクを起床したら，志度タスク起床待ち状態に遷移する．

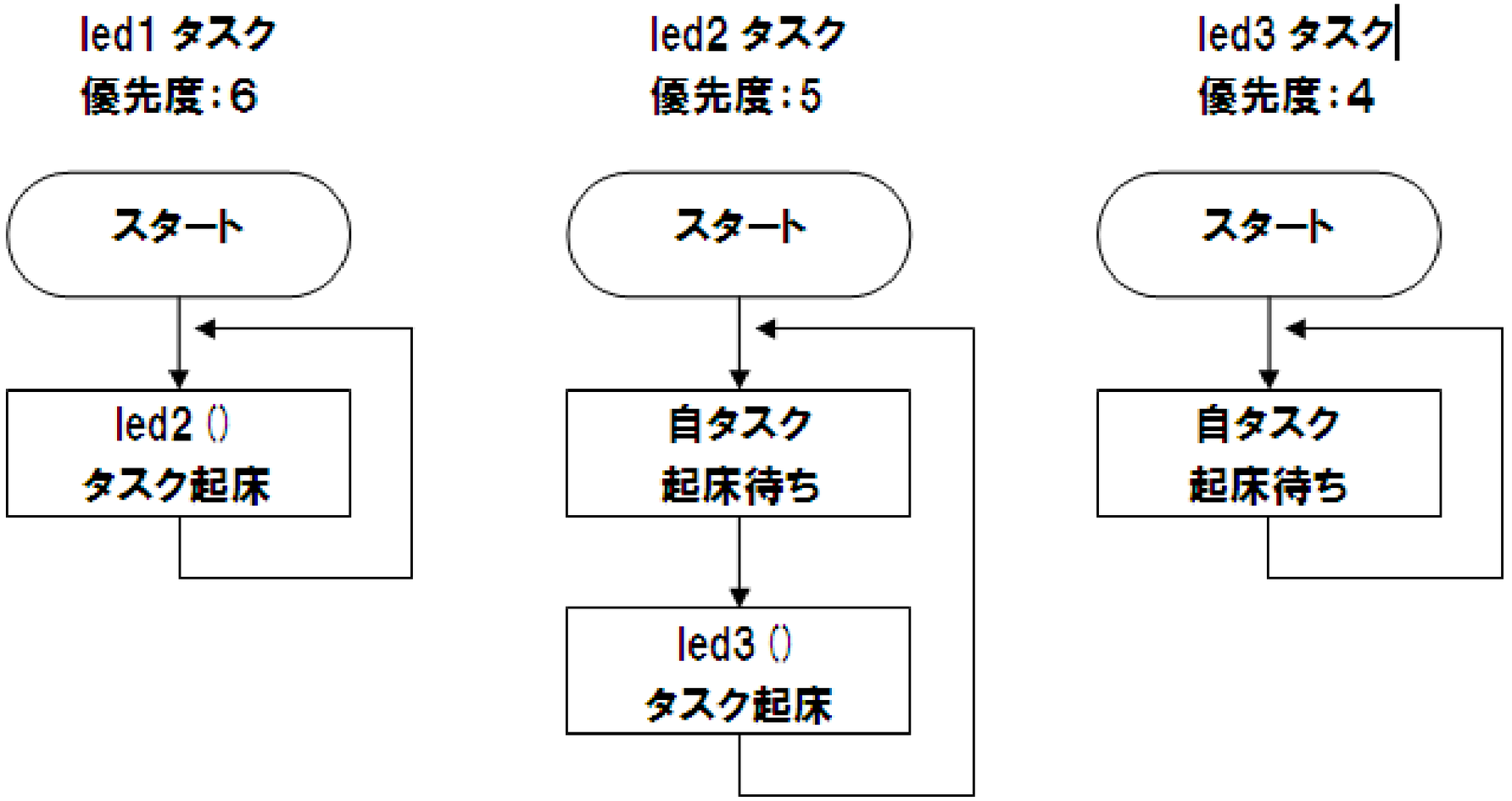


図1．各タスクのフローチャート

【 演習2 タスクチャートを書いてみよう 】

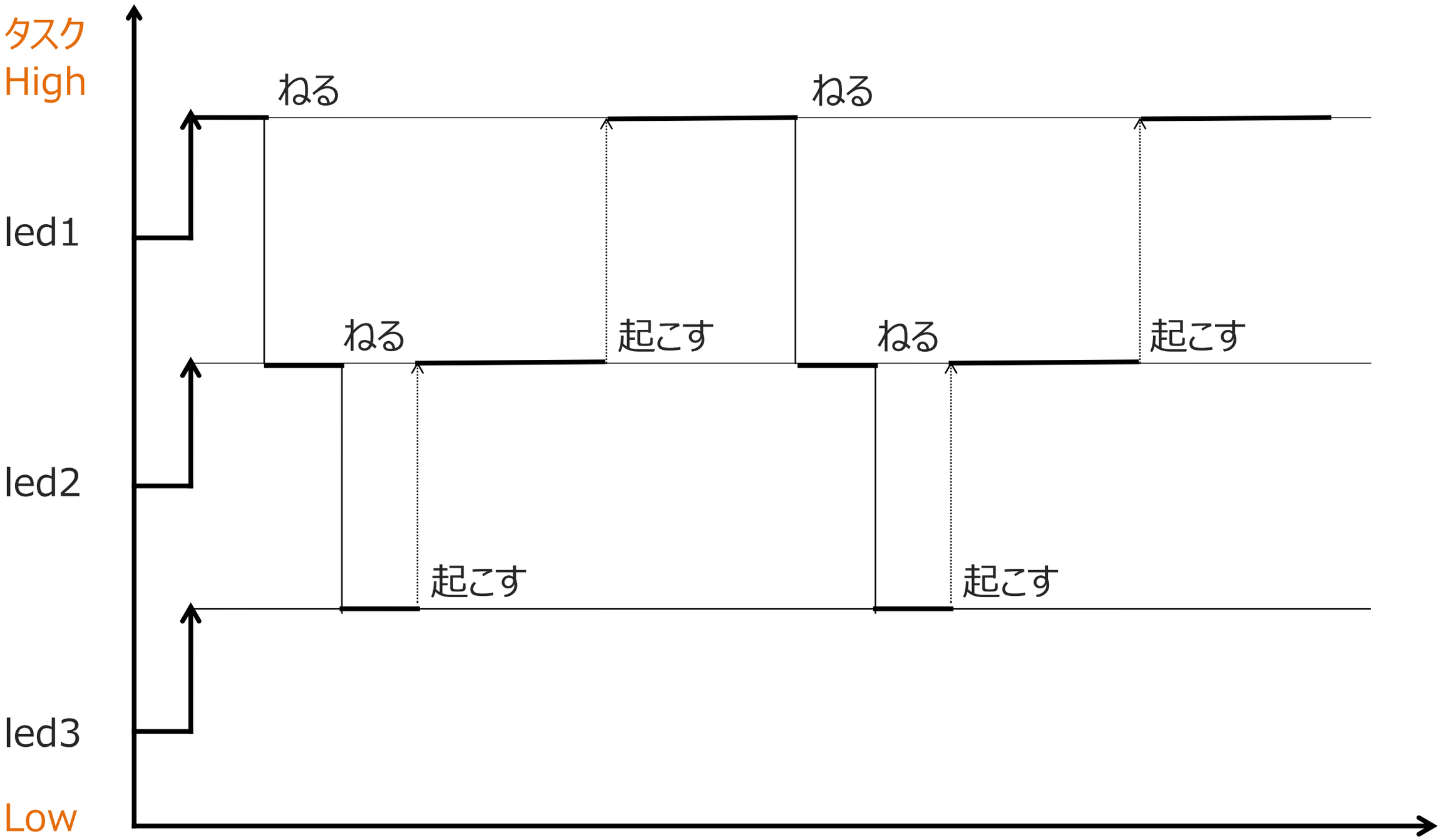


図2. タスクチャート

3. 同期・通信機能

同期・通信機能とは？

- 同期・通信機能は，タスクとは独立したオブジェクトにより，
  タスク間の同期・通信を行うための機能.
  セマフォ，イベントフラグ，メールボックスの機能がある.

セマフォ

資源に対して排他制御や同期を行うためのオブジェクト.

イベントフラグ

イベントの有無をビット毎のフラグで表現することにより，同期を行うため
のオブジェクト.

メールボックス

メモリ上のメッセージを受渡しすることにより，同期と通信を行うため
のオブジェクト.

3. 同期・通信機能「セマフォ」

セマフォ

タスク間での資源(I/Oやメモリ)の排他的使用に利用

システムコール

セマフォ生成 *Create Semaphore*

```
ID semid = tk_cre_sem (T_CSEM *pk_csem);
```

セマフォ削除 *Delete Semaphore*

```
ER ercd = tk_del_sem (ID semid);
```

セマフォ資源返却 *Signal Semaphore*

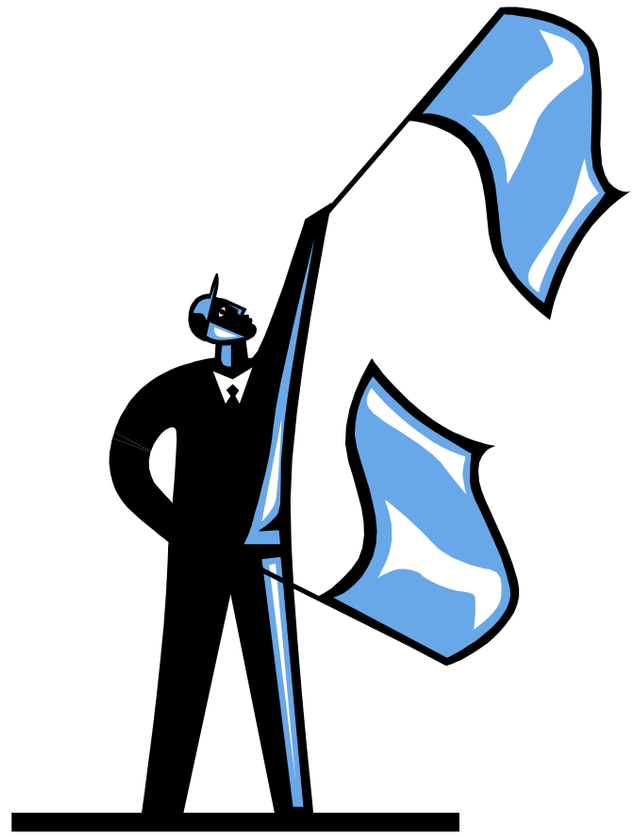
```
ER ercd = tk_sig_sem (ID semid, INT cnt);
```

セマフォ資源獲得 *Wait on Semaphore*

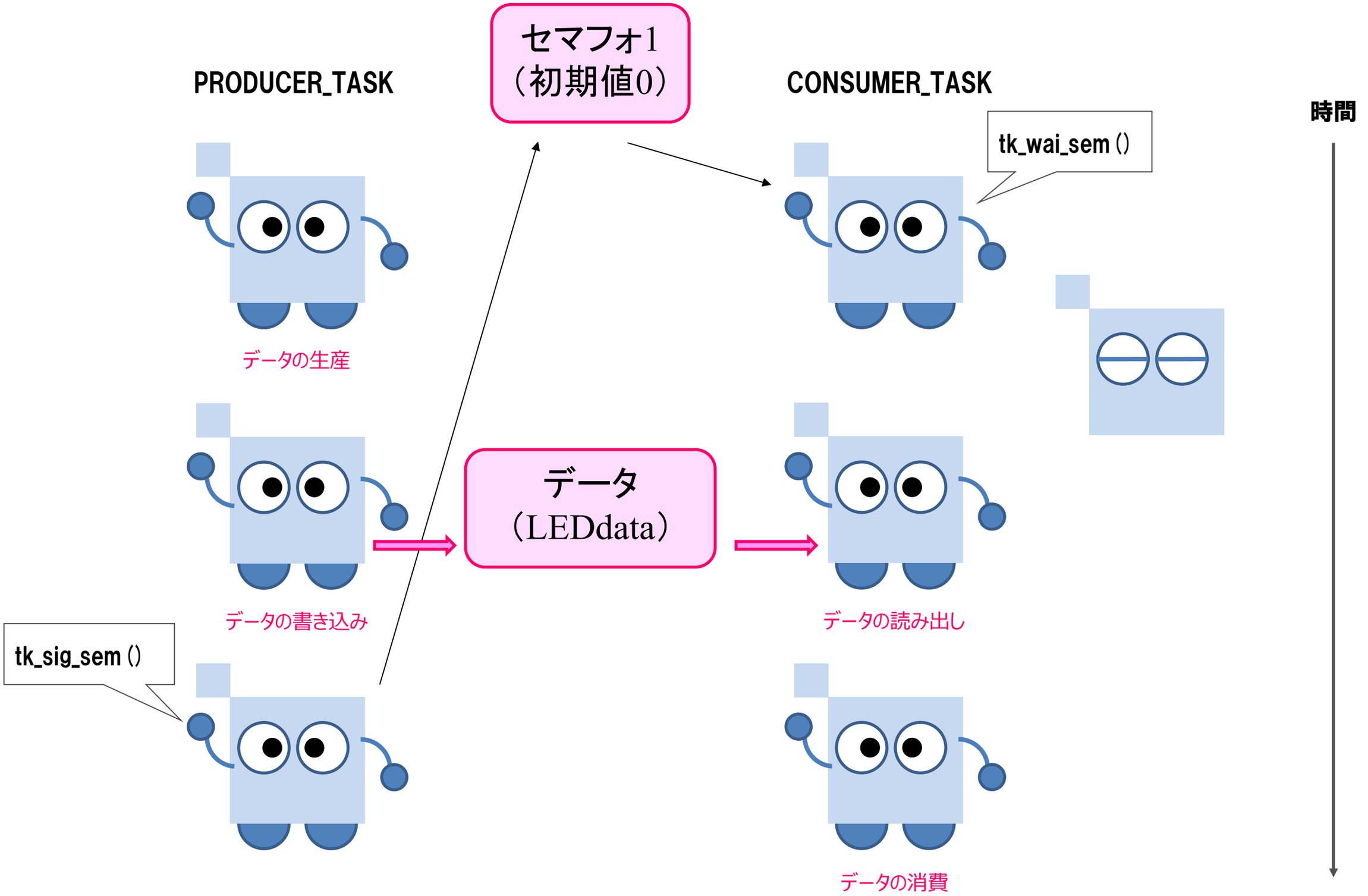
```
ER ercd = tk_wai_sem (ID semid, TMO tmout);
```

セマフォ状態参照 *Refer Semaphore Status*

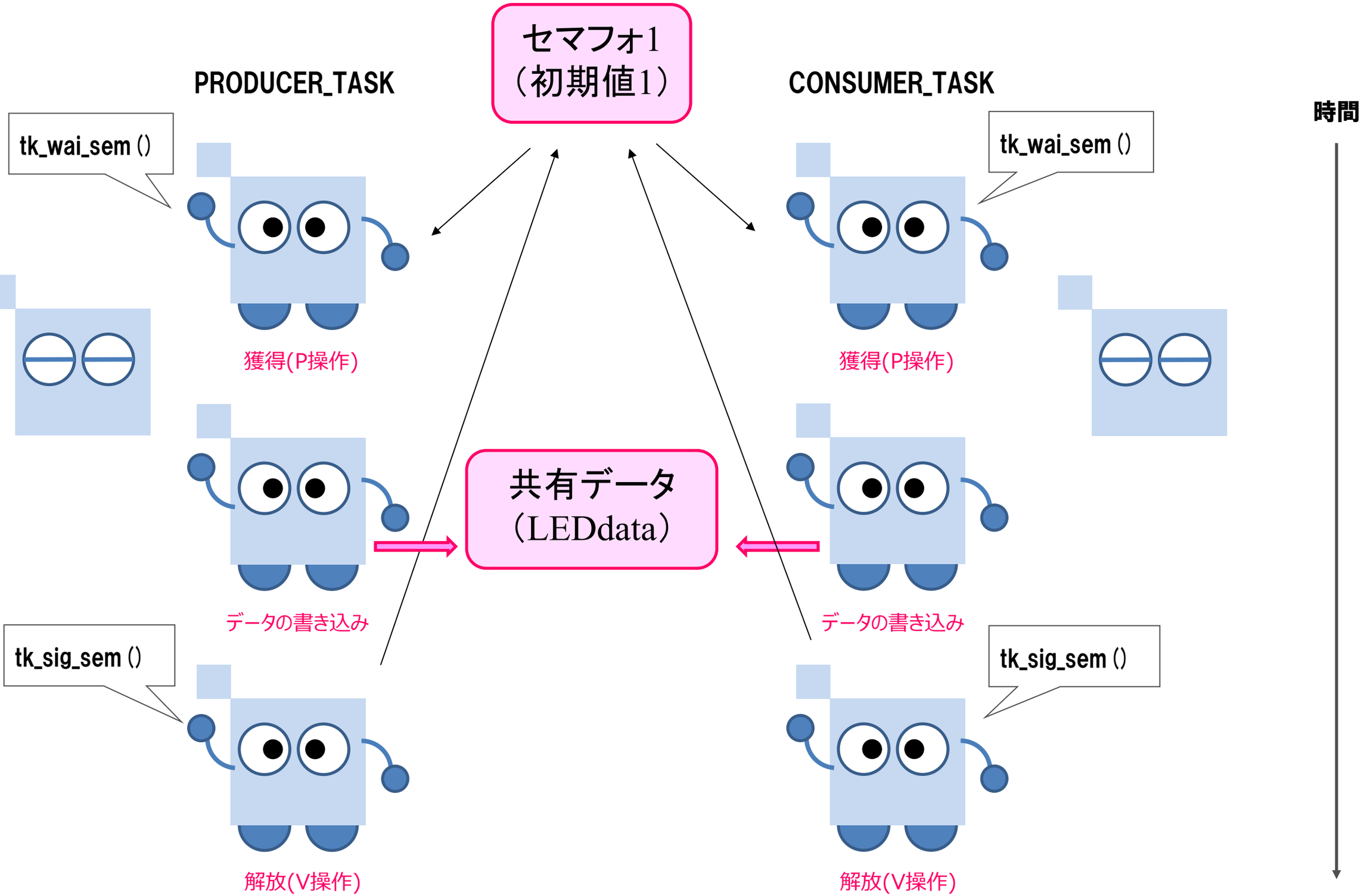
```
ER ercd = tk_ref_sem (ID semid, T_RSEM *pk_rsem);
```



3. 同期・通信機能「セマフォ」：同期



3. 同期・通信機能「セマフォ」：排他



【 演習3 セマフォ 】

■ プロジェクトフォルダ
C:¥ut_realos¥utkernel¥7-M¥uT-Kernel_Samples¥sem

■ プログラム概要
・タスクは，led1(優先度1)，led2(優先度2)，led3(優先度3)，psw1(優先度4)
から構成される
・初期化タスク uinit_task はタスクを動的に生成し，各タスクを順番に起動する
・led1，led2，led3がそれぞれ1つ，2つ，3つのセマフォを要求し，待ち状態に遷移する.
その後，psw1が押されるたびにセマフォが返却され，必要なセマフォ数が確保できるようになったら，led1，led2，led3の待ち状態が解除される.

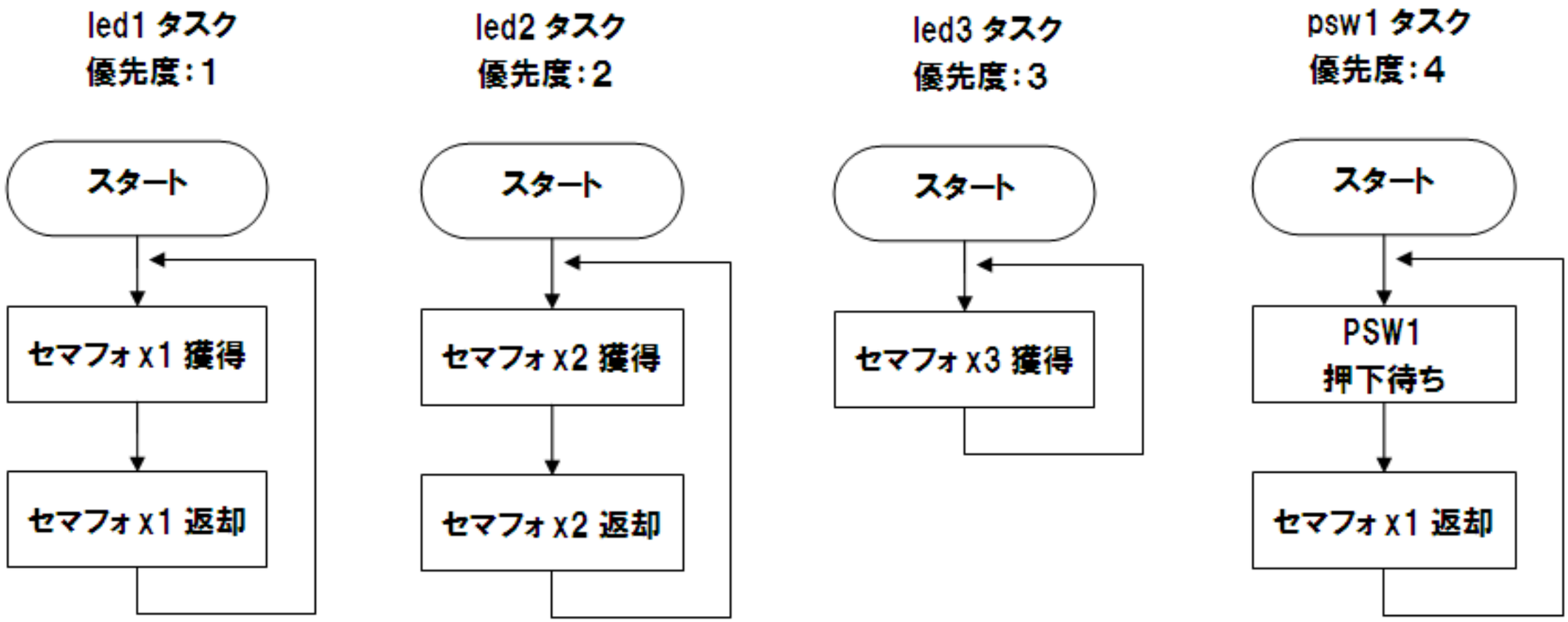


図1. 各タスクのフローチャート

【 演習3 タスクチャートを書いてみよう 】

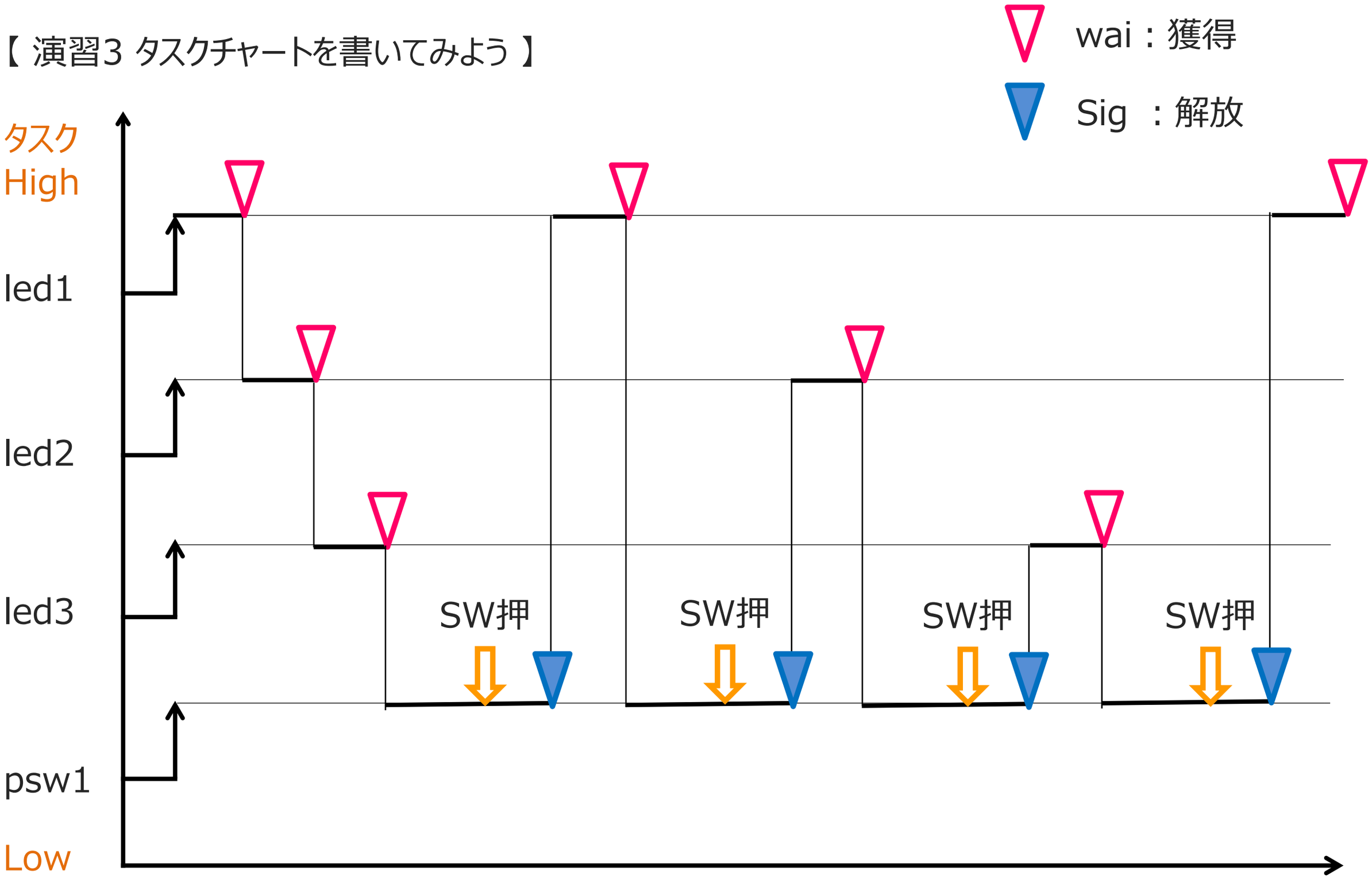
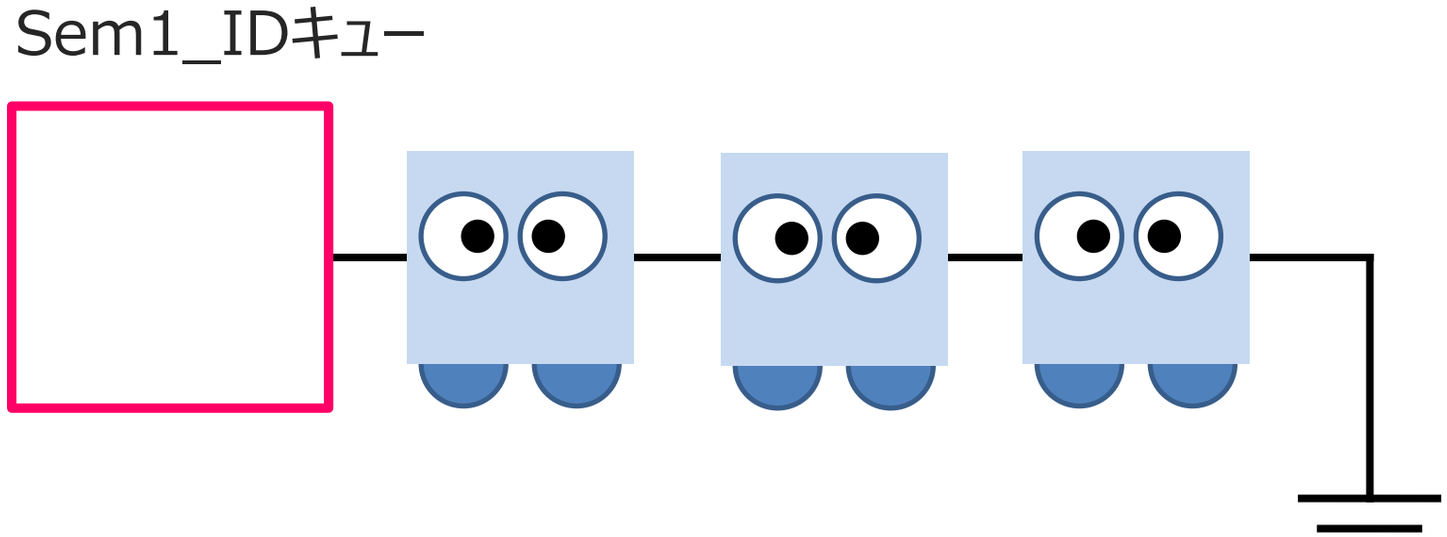


図2. タスクチャート



3. 同期・通信機能「イベントフラグ」

イベントフラグ

イベントの通知をタスク間で通信する

システムコール

イベントフラグ生成 *Create Event Flag*

```
ID flgid = tk_cre_flg (T_CFLG *pk_cflg);
```

イベントフラグ削除 *Delete Event Flag*

```
ER ercd = tk_del_flg (ID flgid);
```

イベントフラグのセット *Set Event Flag*

```
ER ercd = tk_set_flg (ID flgid, UINT setptn);
```

イベントフラグのクリア *Clear Event Flag*

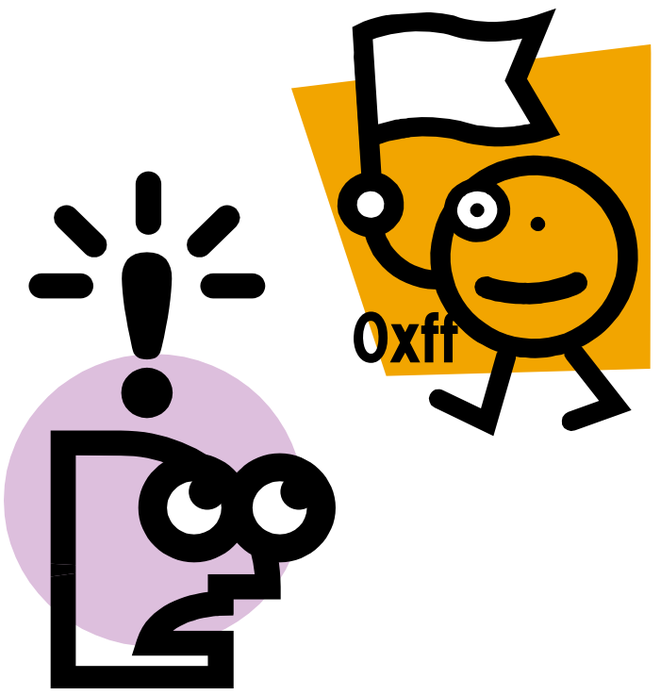
```
ER ercd = tk_clr_flg (ID flgid, UINT clrptn);
```

イベントフラグの待ち *Wait Event Flag*

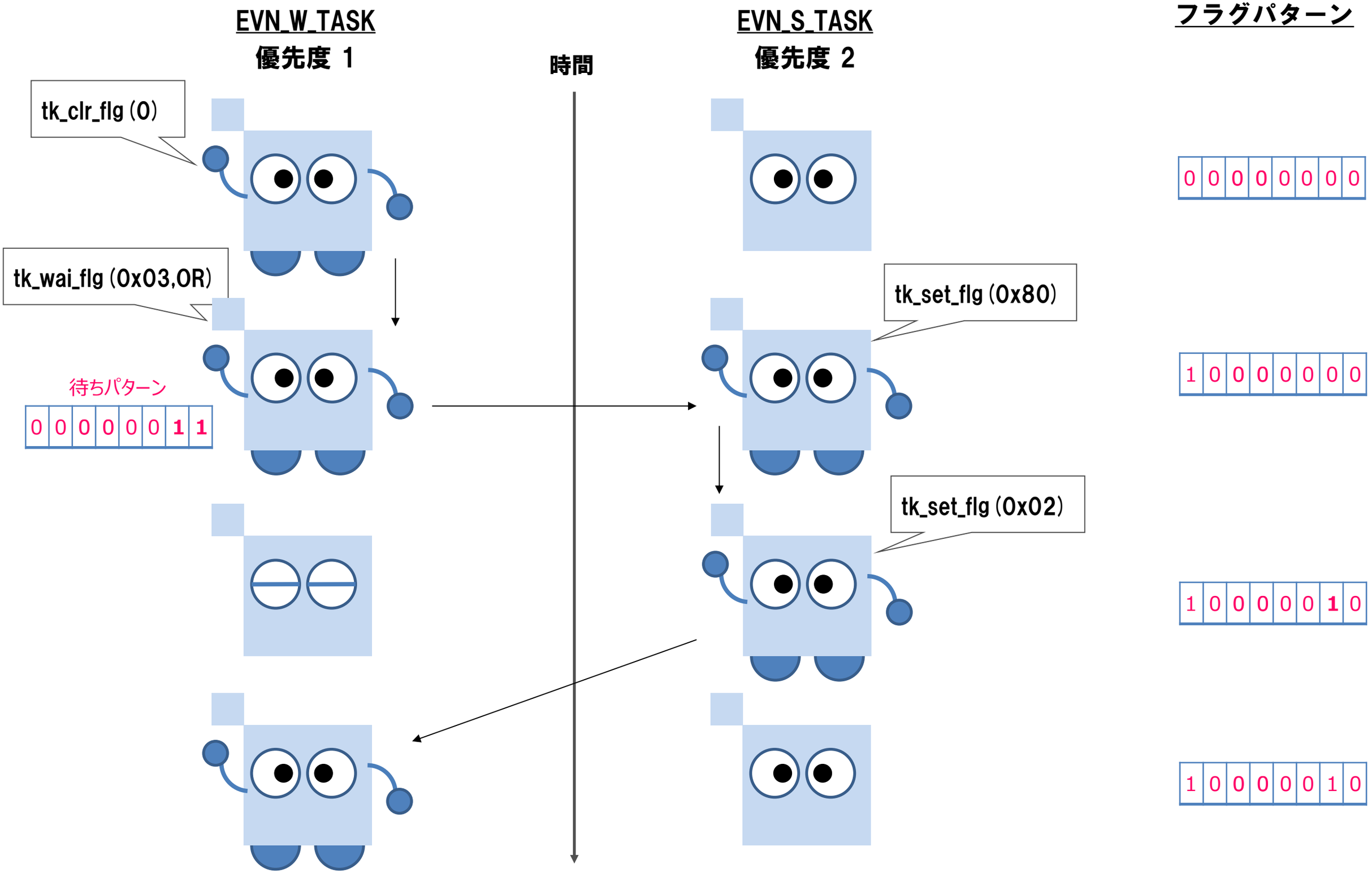
```
ER ercd = tk_wai_flg (ID flgid, UINT waiptn,
UINT wfmode, UINT *p_flgptn, TMO tmout);
```

イベントフラグ状態参照 *Refer EventFlag Status*

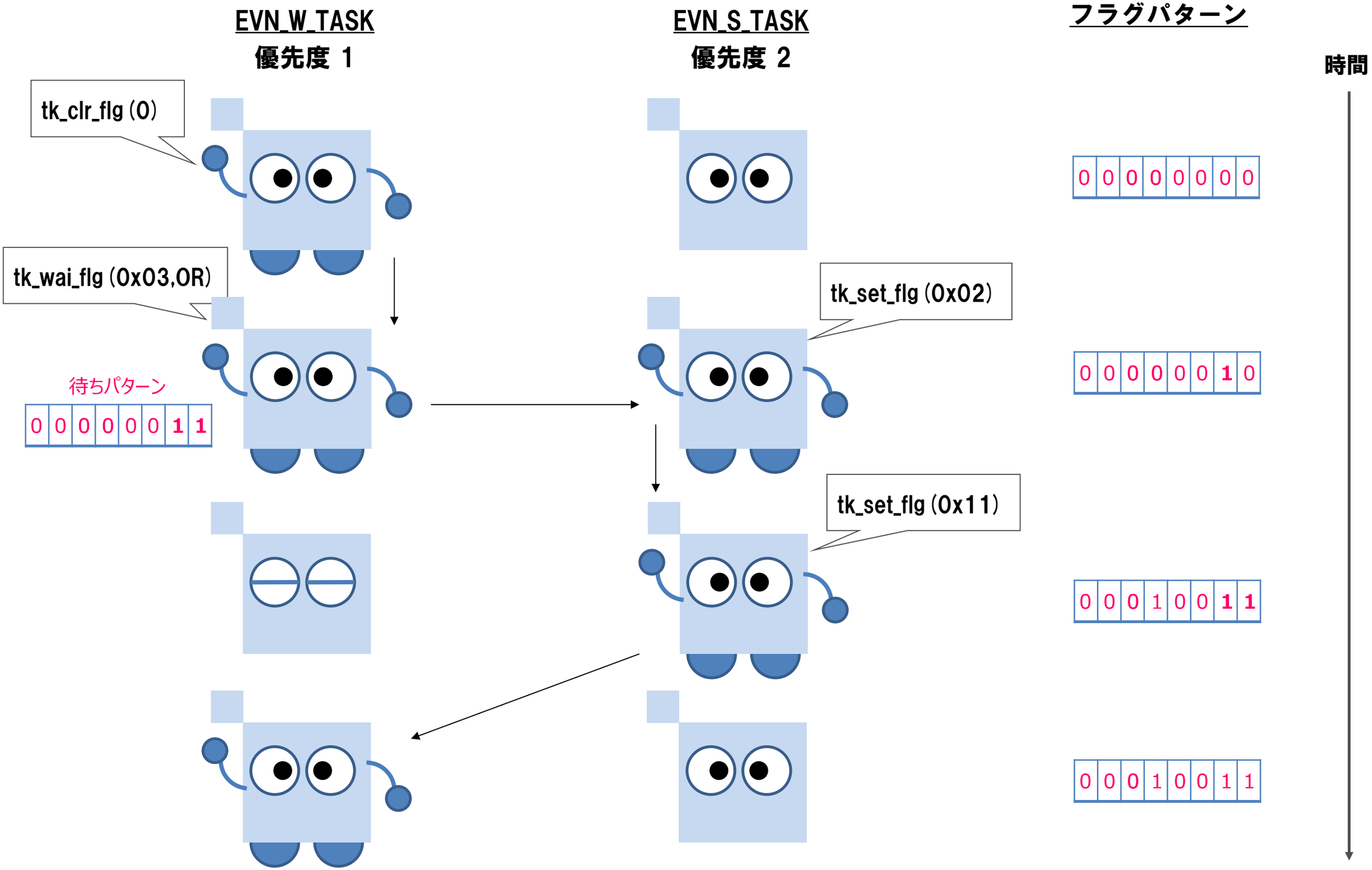
```
ER ercd = tk_ref_flg (ID flgid, T_RFLG *pk_rflg);
```



3. 同期・通信機能「イベントフラグ」:OR



3. 同期・通信機能「イベントフラグ」:AND



【 演習4 イベントフラグ 】

■ プロジェクトフォルダ
C:¥ut_realos¥utkernel¥7-M¥uT-Kernel_Samples¥ev_flag

■ プログラム概要
・タスクは，led1(優先度1)，psw1(優先度2) から構成される
・初期化タスク uinit_task はタスクを動的に生成し，各タスクを順番に起動する
・psw1タスクはpsw1スイッチの状態をサンプリングし，psw1スイッチが押されたらフラグパターンを0x00から増加させる（0x00，0x01，0x02，0x03，・・・）
・led1は要求するフラグパターンを設定し，パターンが実行されるまで待状態に遷移する.
psw1によって発行されるパターンが要求パターンと一致すれば，待ち状態が解除される.

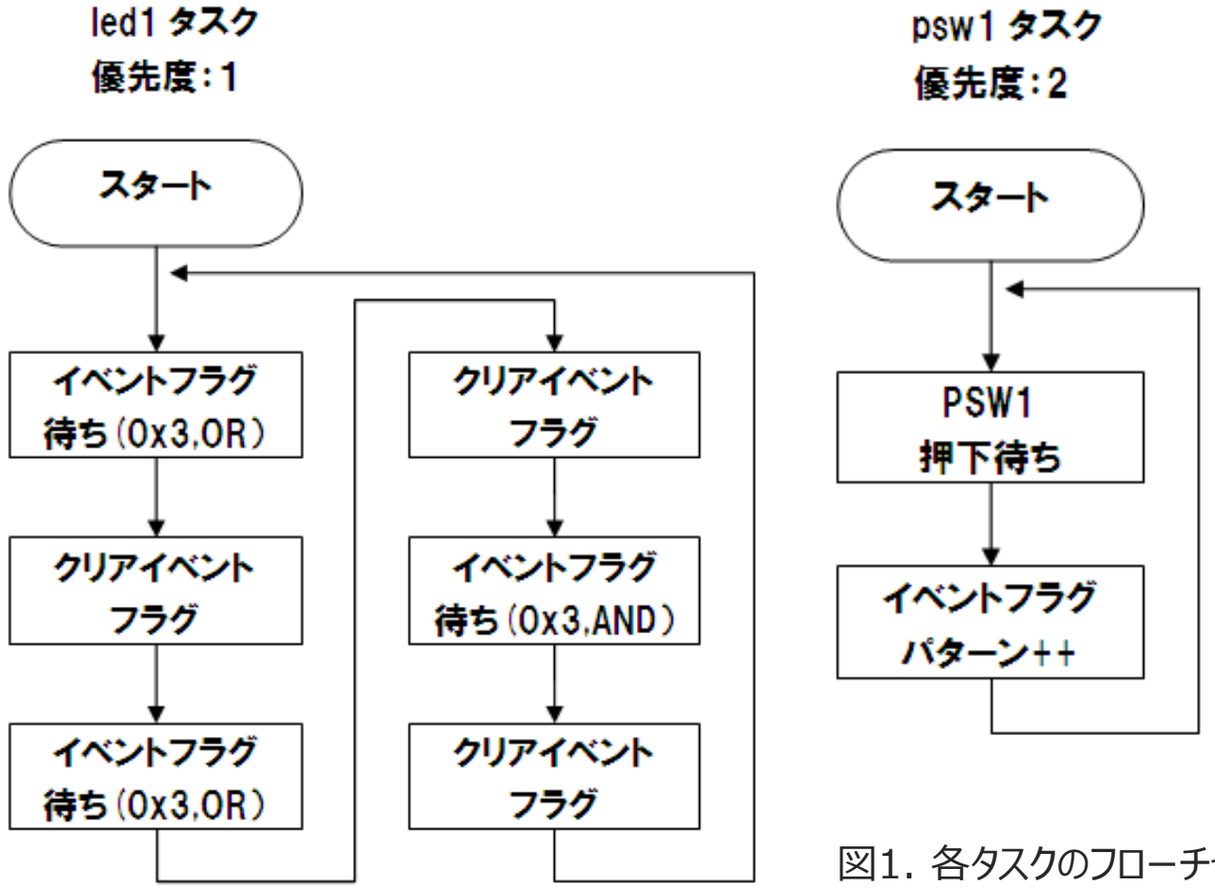
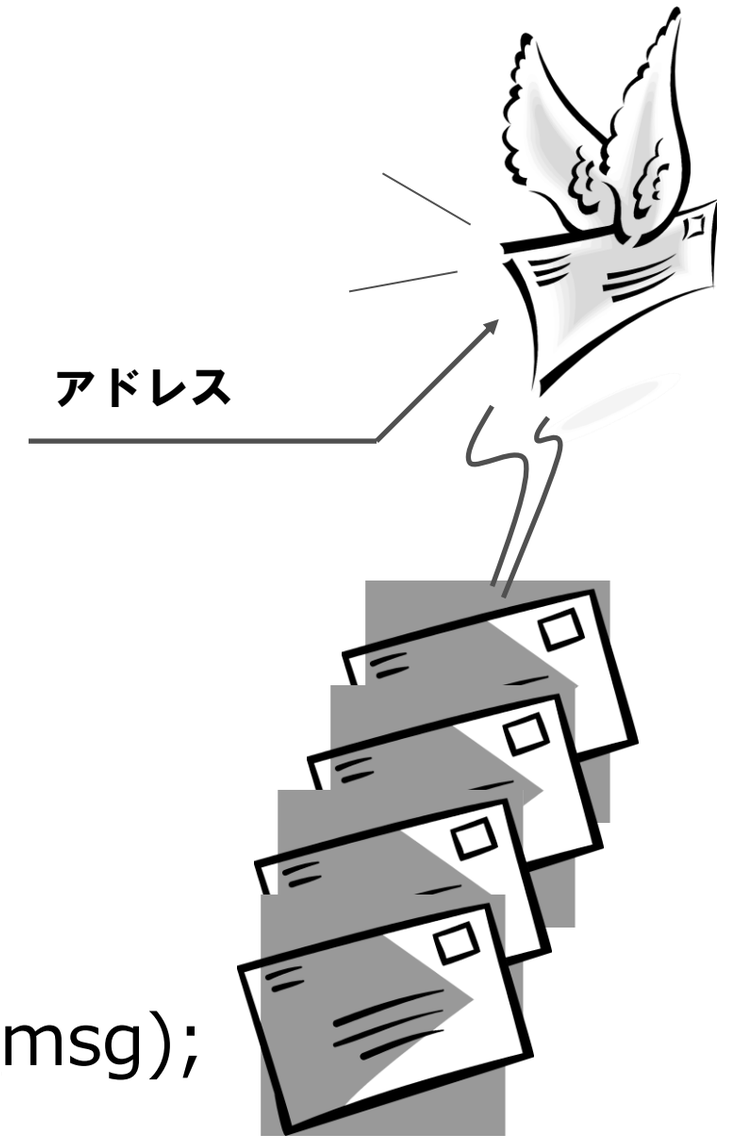


図1．各タスクのフローチャート

3. 同期・通信機能「メールボックス」

メールボックス

タスク間でメッセージ領域のアドレスを通信



システムコール

メールボックス生成 *Create Mailbox*

```
ID mbxid = tk_cre_mbx (T_CMBX *pk_cmbx);
```

メールボックス削除 *Delete Mailbox*

```
ER ercd = tk_del_flg (ID mbxid);
```

メールボックスへ送信

```
ER ercd = tk_snd_mbx (ID mbxid, T_MSG *pk_msg);
```

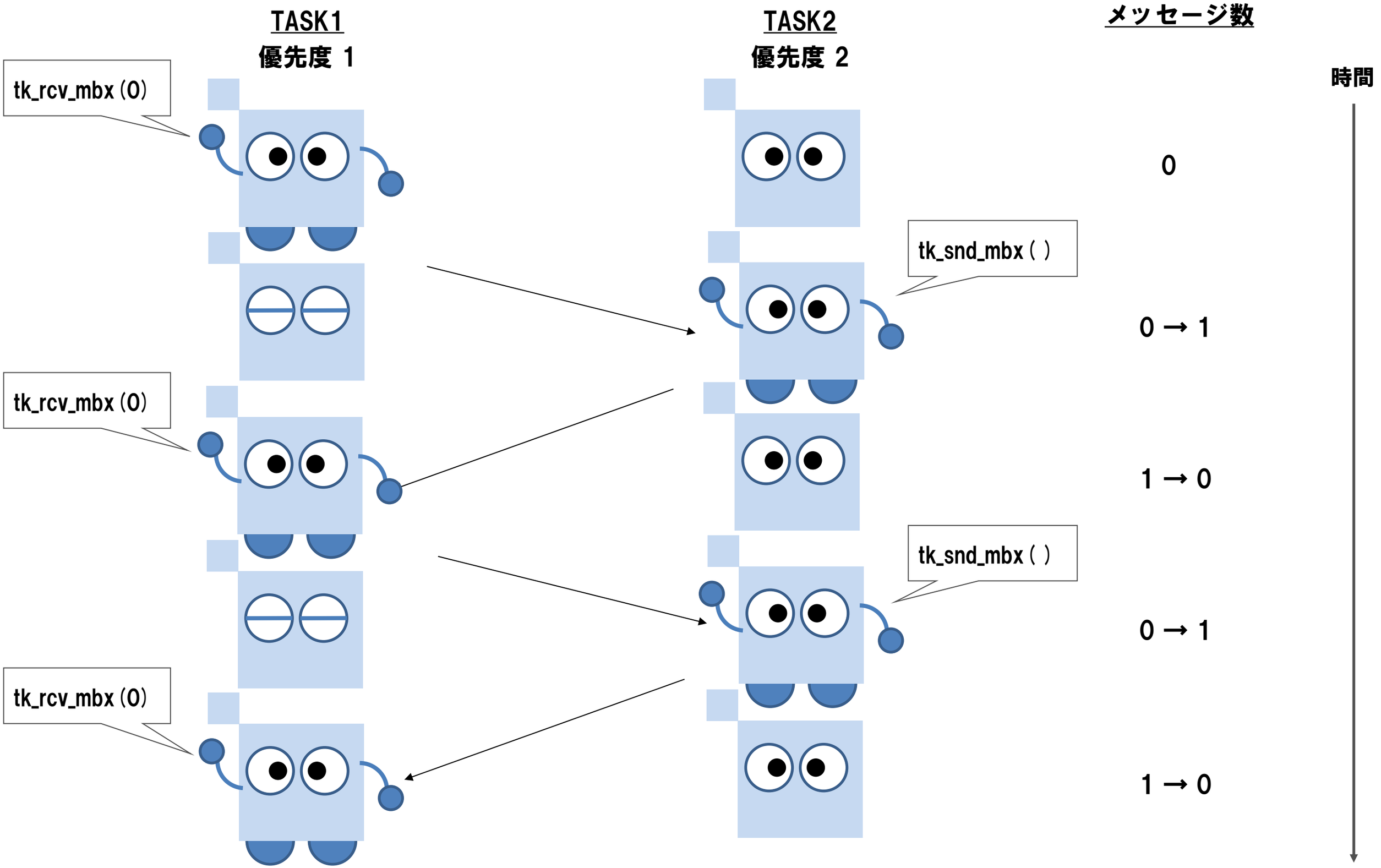
メールボックスから受信

```
ER ercd = tk_rcv_mbx (ID mbxid, TMSG **ppk_msg, TMO tmout);
```

メールボックス状態参照

```
ER ercd = tk_ref_mbx (ID mbix, T_RMBX *pk_rmbx);
```

3. 同期・通信機能「メールボックス」



【 演習5 メールボックス 】

■ プロジェクトフォルダ
C:¥ut_realos¥utkernel¥7-M¥uT-Kernel_Samples¥mailbox

■ プログラム概要
・タスクは，rcv1(優先度1)，snd1(優先度2) から構成される
・初期化タスク uinit_task はタスクを動的に生成し，各タスクを順番に起動する
・rcv1が実行状態になったら受信できるメッセージが発生するまで待ちます.
そこで，snd1からメッセージが送信されたら，rcv1の待ち状態が解除され，
rcv1が受信したメッセージがLCDとターミナルI/Oに表示される.

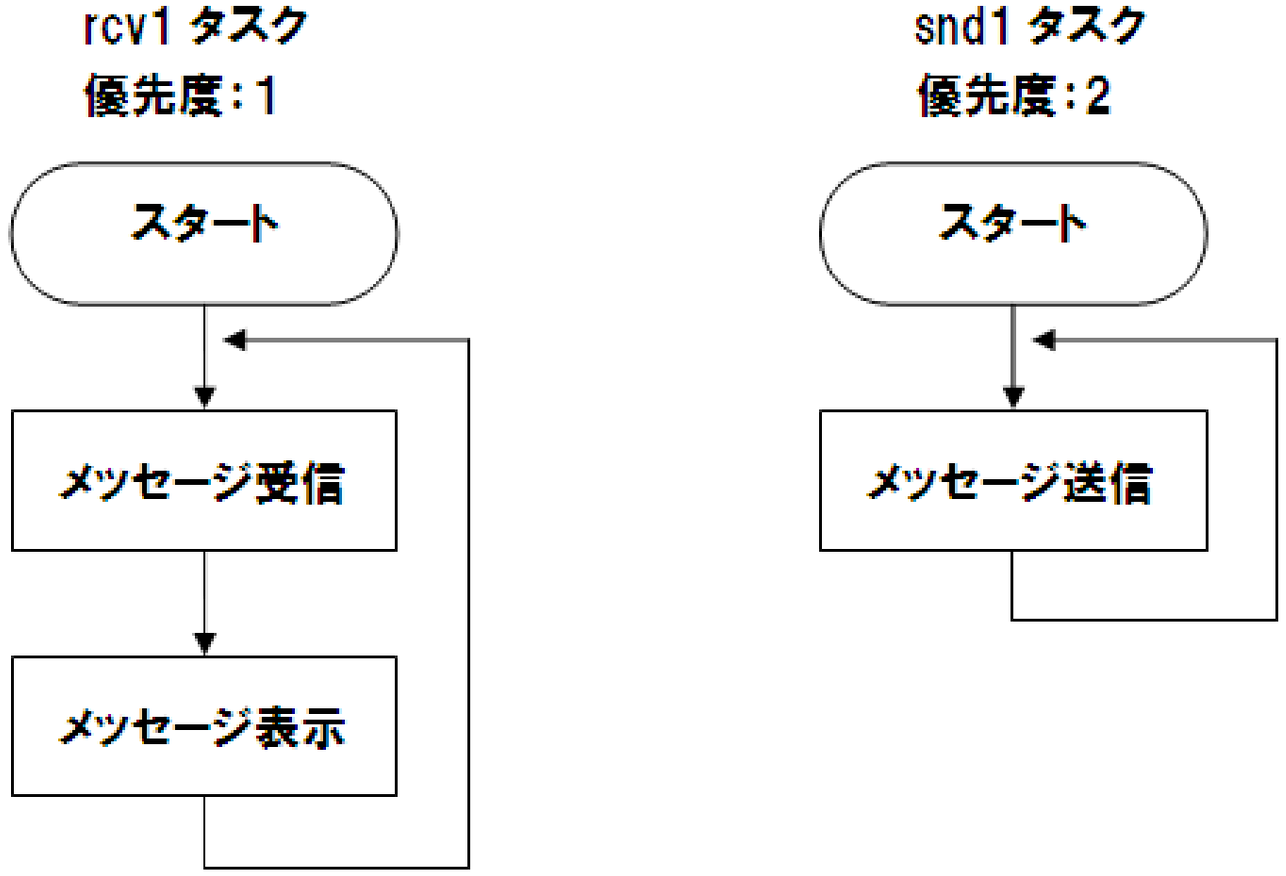


図1．各タスクのフローチャート

4. 割込み管理機能

割込み管理機能とは？

- 割込み管理機能とは，外部割込み及びCPU例外に対するハンドラの定義や割込み制御などの操作を行う機能である.
- 割込みハンドラは，タスク独立部として扱われる.

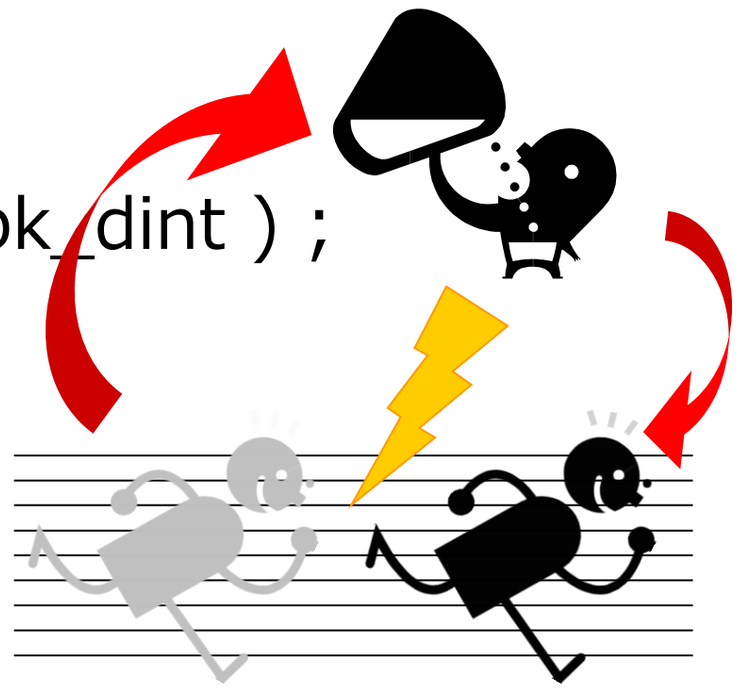
システムコール

割込みハンドラの定義

```
ER ercd = tk_def_int ( UINT dintno, T_DINT *pk_dint ) ;
```

割込みハンドラから復帰

```
void tk_ret_int ( void ) ;
```



【 演習6 割込みハンドラ 】

■ プロジェクトフォルダ
C:¥ut_realos¥utkernel¥7-M¥uT-Kernel_Samples¥int

■ プログラム概要
・タスクは，led1(優先度1)，led2(優先度2) と割込みハンドラ int1 から構成される
・初期化タスク uinit_task は割込みハンドラ int1 の登録を行った後，
タスクを動的に生成し，各タスクを順番に起動する
・led1とled2が実行状態からすぐに起床待ち状態に遷移する.
psw1またはpsw2が押されたら，割込みハンドラが起動し，led1またはled2を起床する.
その後，起床されたタスクが該当するLEDを点灯/消灯し，再度起床待ち状態に遷移する.

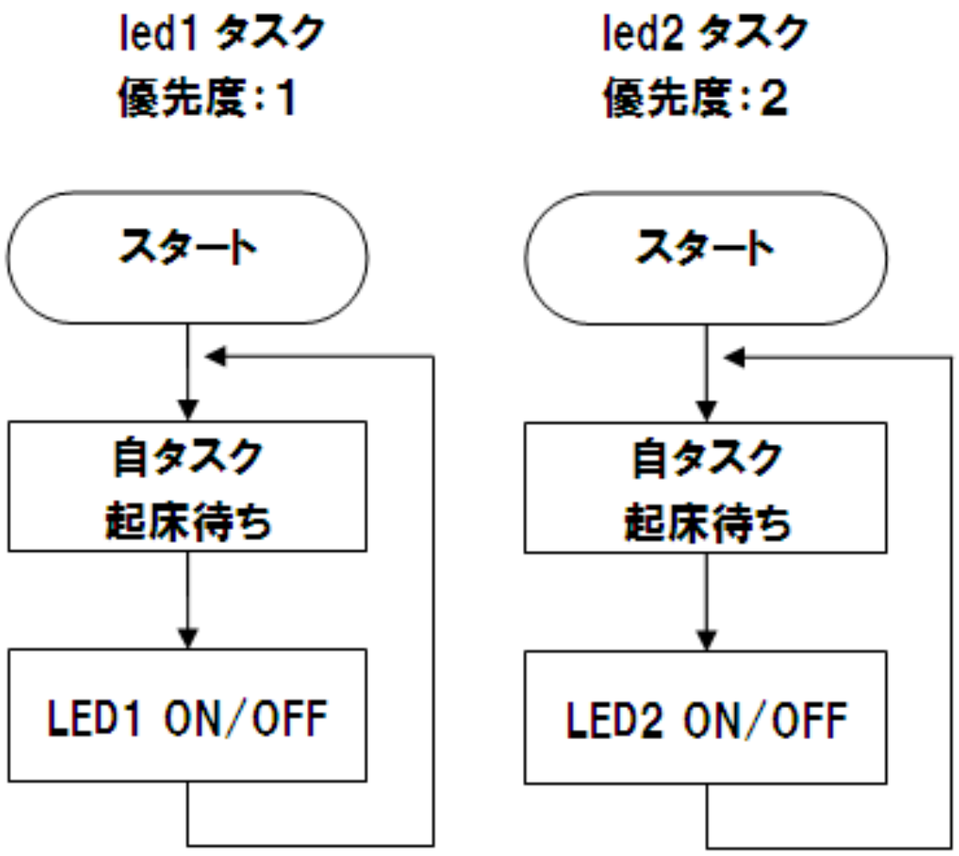
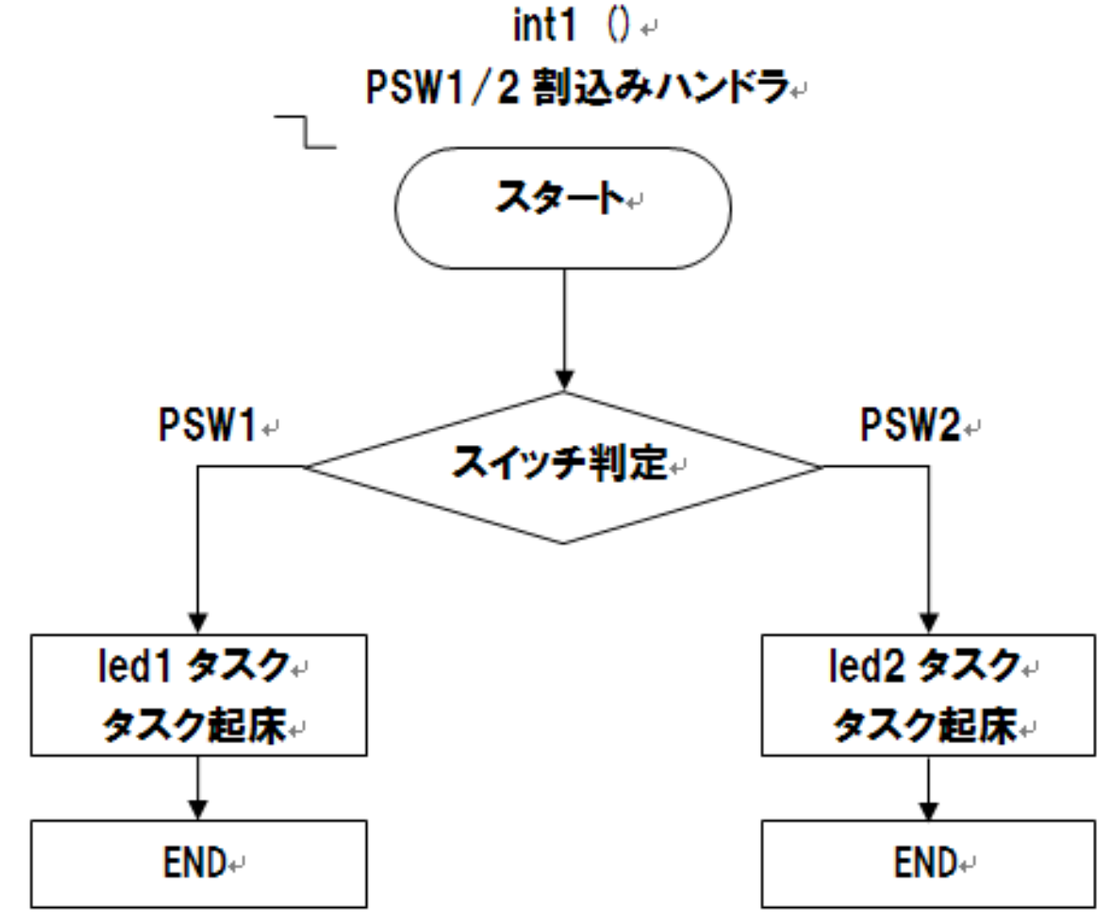


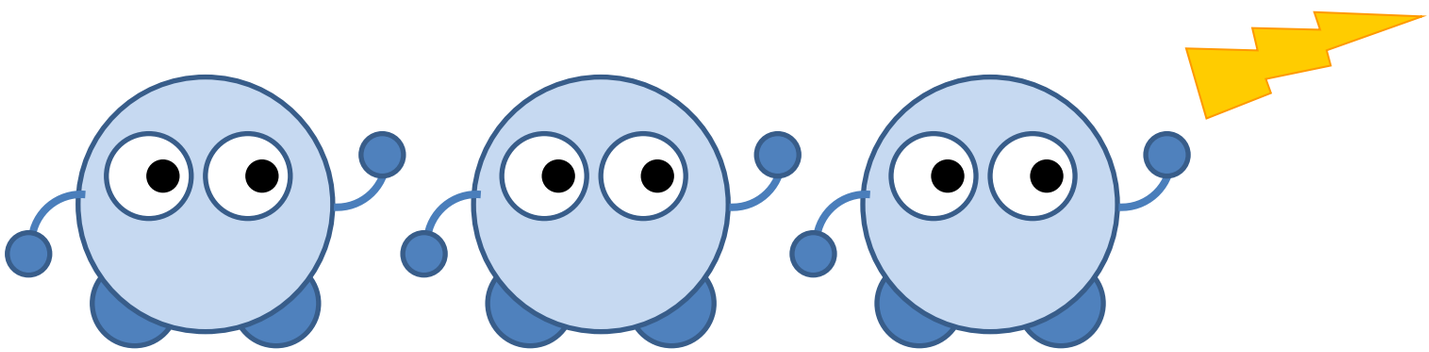
図1. 各タスクのフローチャート



5. 時間管理機能

時間管理機能とは？

- 時間管理機能とは，時間に依存した処理を行うための機能.
- 「システム時刻管理」，「周期ハンドラ」，「アラームハンドラ」の各機能が含まれる.



5. 時間管理機能「システム時刻管理」

システム時刻管理機能

システム時刻管理機能とは，システム時刻を操作するための機能.

システムコール

システム時刻設定 *Set Time*

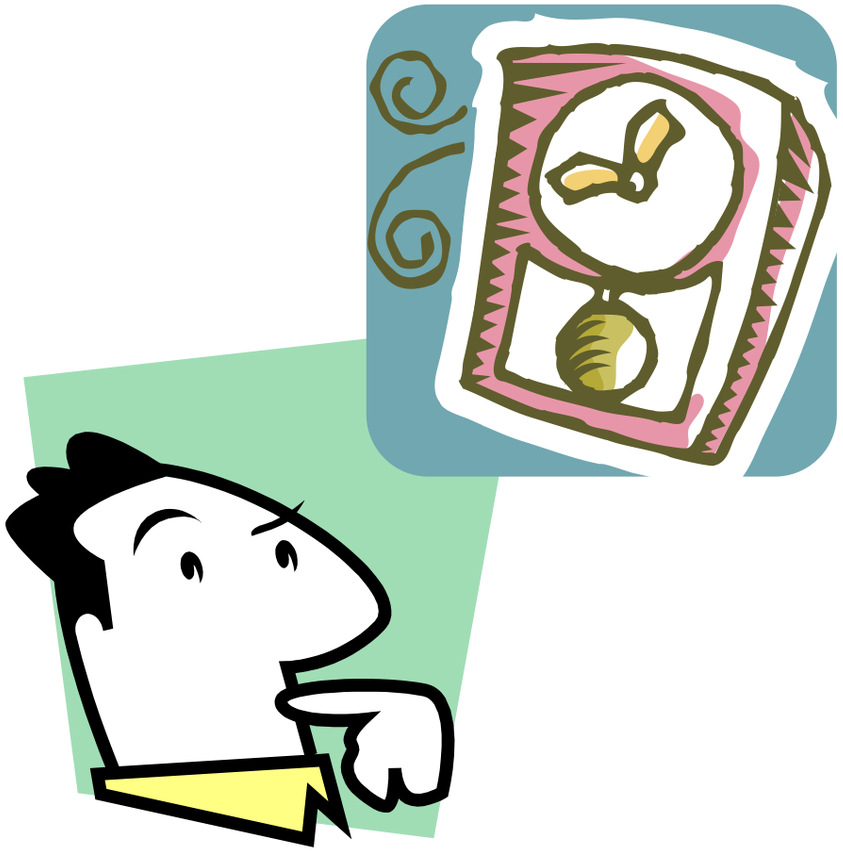
```
ER ercd = tk_set_tim (SYSTIM *pk_tim);
```

システム時刻参照 *Get Time*

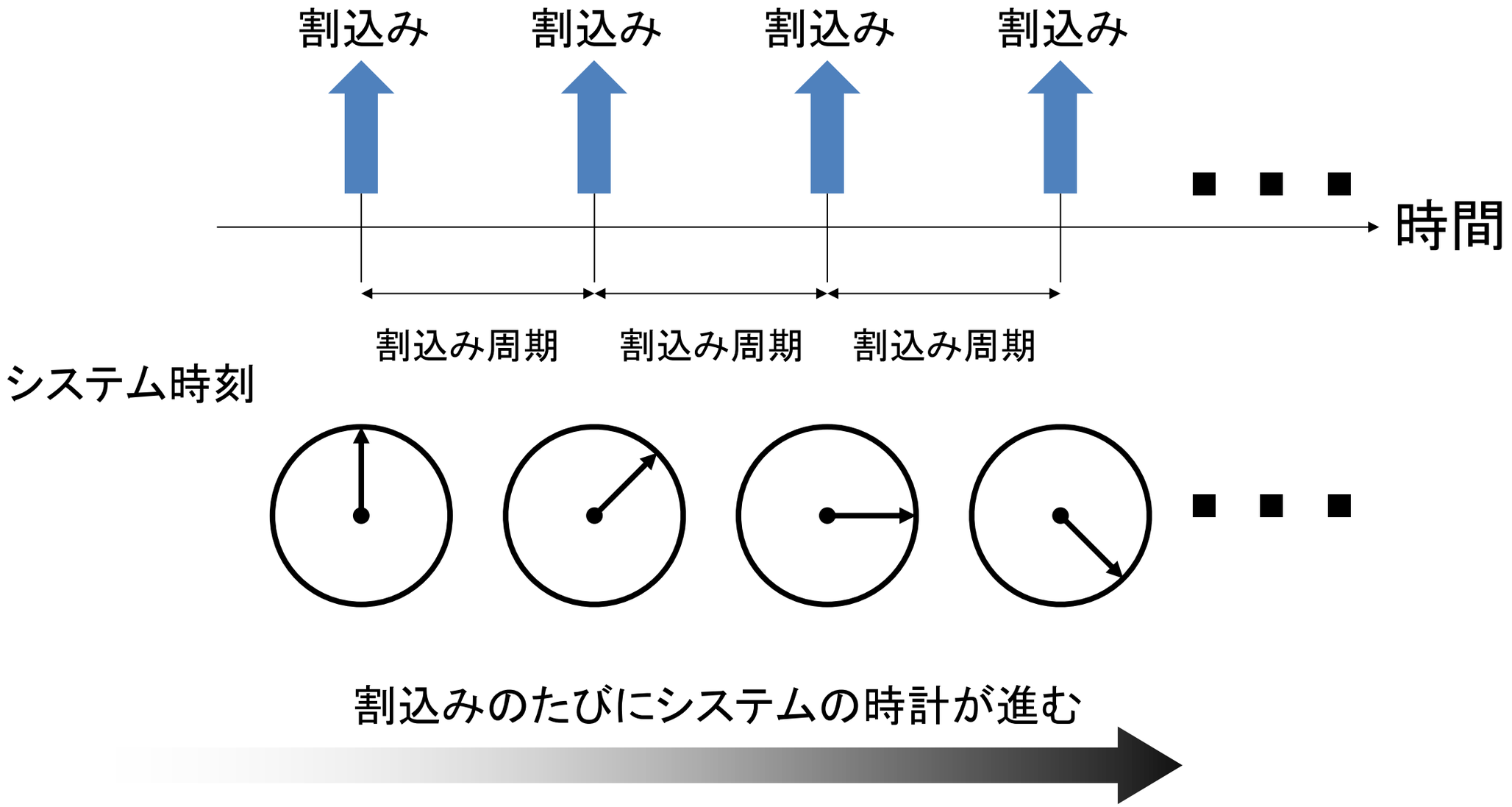
```
ER ercd = tk_get_tim (SYSTIM *pk_tim);
```

システム稼動時間参照

```
ER ercd = tk_get_otm (SYSTIM *pk_tim);
```



5. 時間管理機能「システム時刻管理」



5. 時間管理機能「システム時刻管理」:補足

◆ 相対時間とシステム時刻

● 相対時間（RELTIM型）

➢ イベントの発生する時刻を，システムコールを呼び出した時刻などからの相対値で
指定する場合には，相対時間（RELTIM型）を用いる.
相対時間を用いてイベントの発生時刻が指定された場合，イベントの処理は，
基準となる時刻から指定された以上の時間が経過した後に行うことを保証しなければならない.

● システム時刻（SYSTIM型）

➢ 時刻を絶対値で指定する場合

※ RELTIM，TMOで指定された時間は，指定された時間以上経過した後に
タイムアウト等が起こることを保証しなければならない.

5. 時間管理機能「周期ハンドラ」

周期ハンドラ

周期ハンドラは，一定周期で起動される タイムイベントハンドラ.

システムコール

周期ハンドラの生成 *Create Cyclic Handler*

```
ID cycid = tk_cre_cyc (T_CCYC *pk_ccyc);
```

周期ハンドラの削除 *Delete Cyclic Handler*

```
ER ercd = tk_del_cyc (ID cycid);
```

周期ハンドラの動作開始 *Start Cyclic Handler*

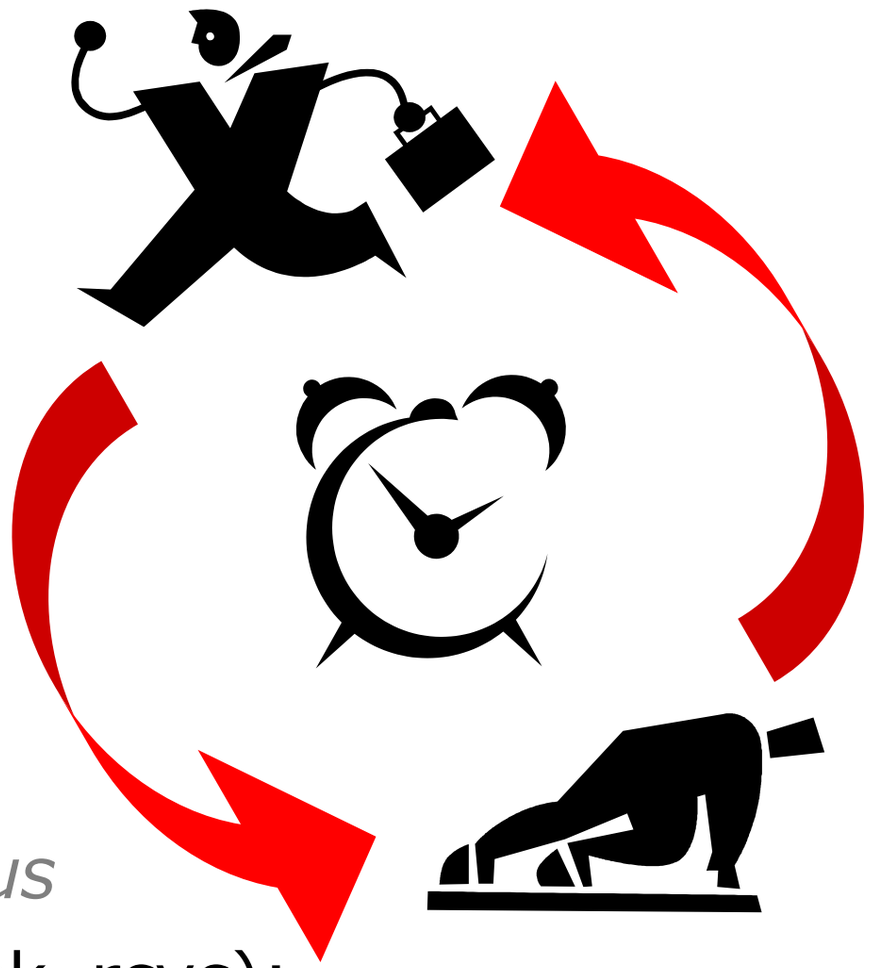
```
ER ercd = tk_sta_cyc (ID cycid);
```

周期ハンドラの動作停止 *Stop Cyclic Handler*

```
ER ercd = tk_stp_cyc (ID cycid);
```

周期ハンドラ状態参照 *Refer Cyclic Handler Status*

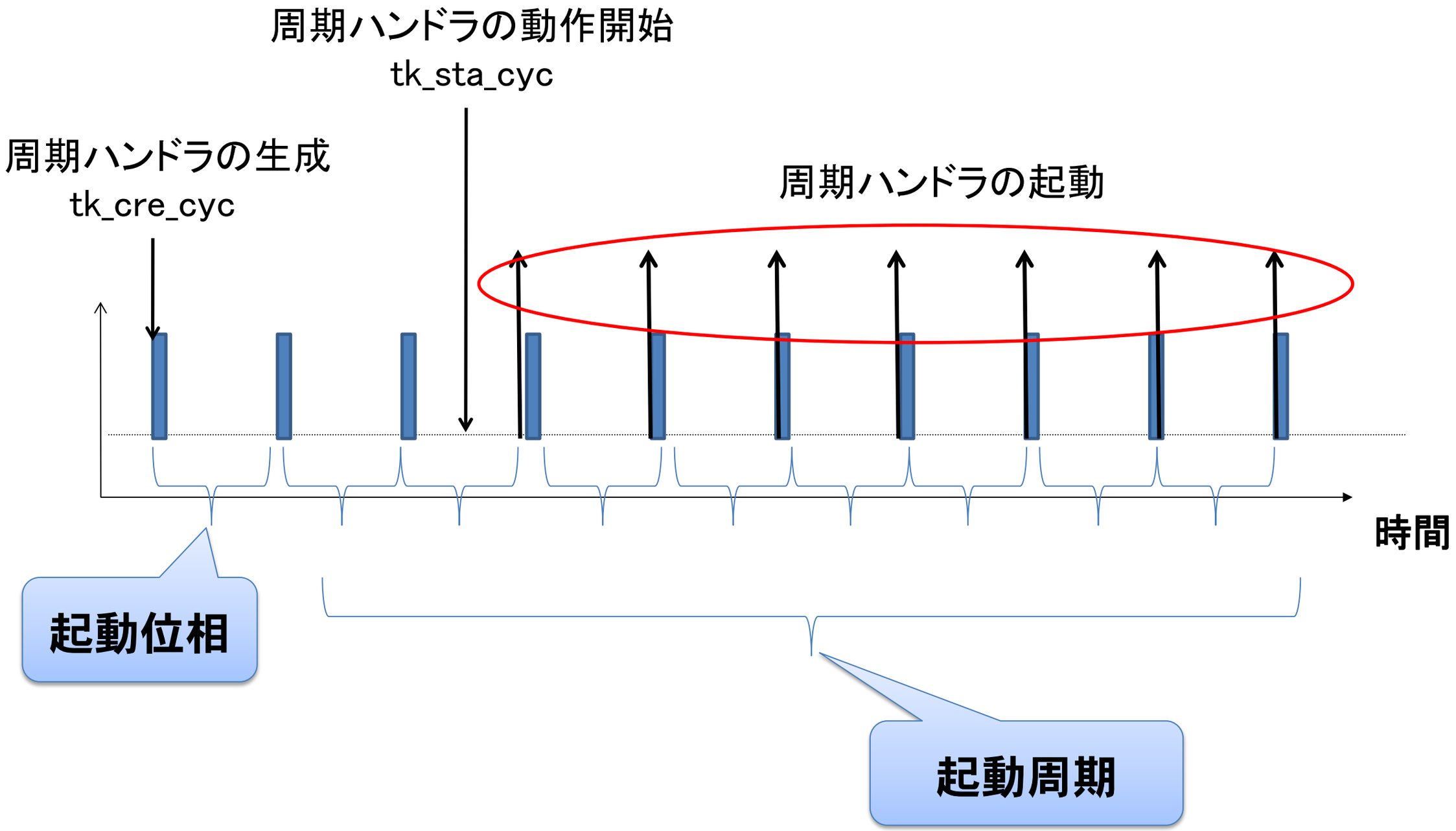
```
ER ercd = tk_ref_cyc (ID cycid, T_RCYC *pk_rcyc);
```



5. 時間管理機能「周期ハンドラ」:注意点

起動位相を保存する場合

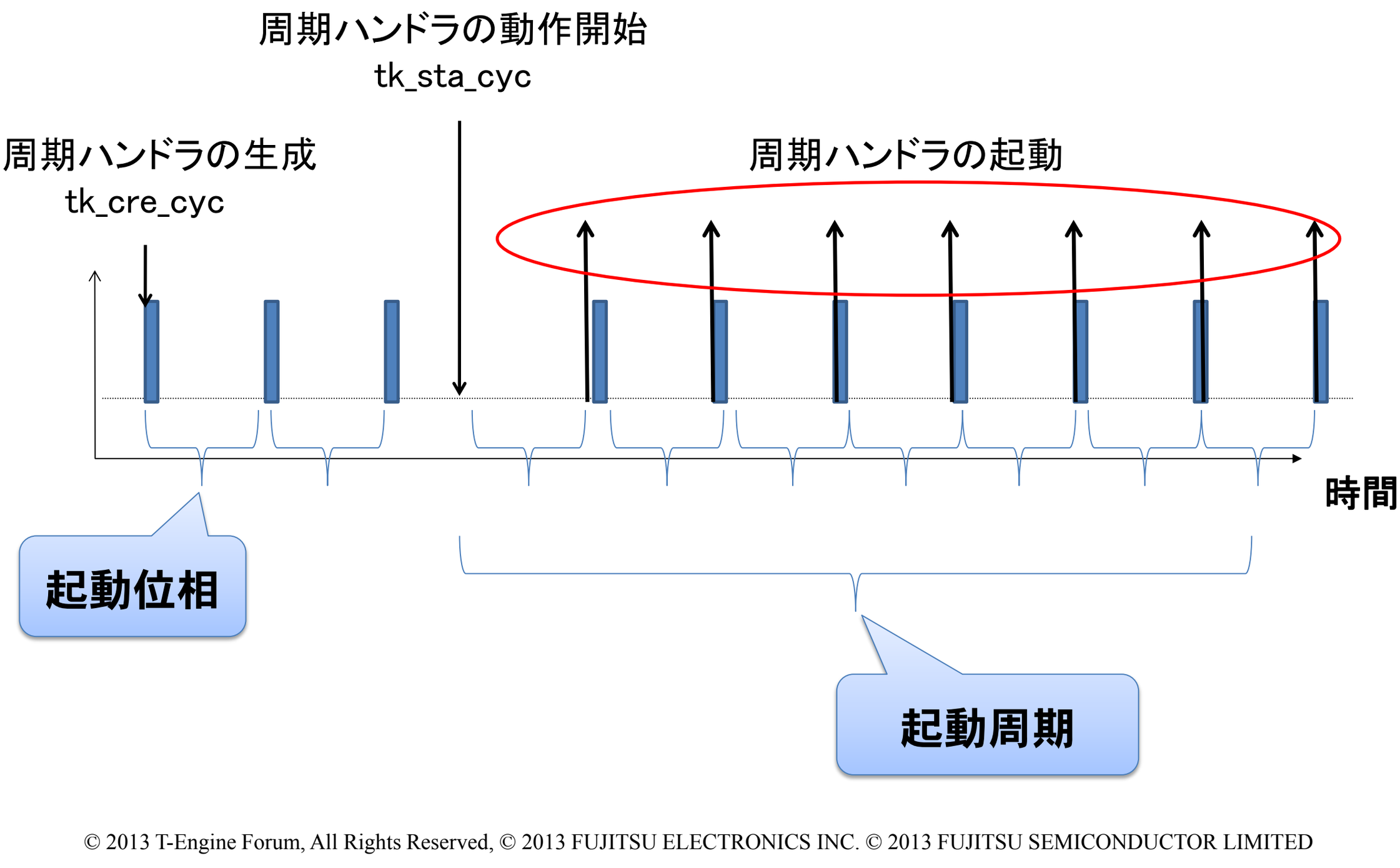
※ tk_cre_cyc の属性として TA_PHS を指定



5. 時間管理機能「周期ハンドラ」:注意点

起動位相を保存しない場合

※ tk_cre_cyc の属性として TA_PHS を指定しない



5. 時間管理機能「アラームハンドラ」

アラームハンドラ

指定した時刻に起動されるタイムイベントハンドラ.

代表的なシステムコール

アラームハンドラの生成 *Create Alarm Handler*

```
ID almid = tk_cre_alm (T_CALM *pk_calm);
```

アラームハンドラの削除 *Delete Alarm Handler*

```
ER ercd = tk_del_alm (ID almid);
```

アラームハンドラの動作開始 *Start Alarm Handler*

```
ER ercd = tk_sta_alm (ID almid, RELTIM almtim);
```

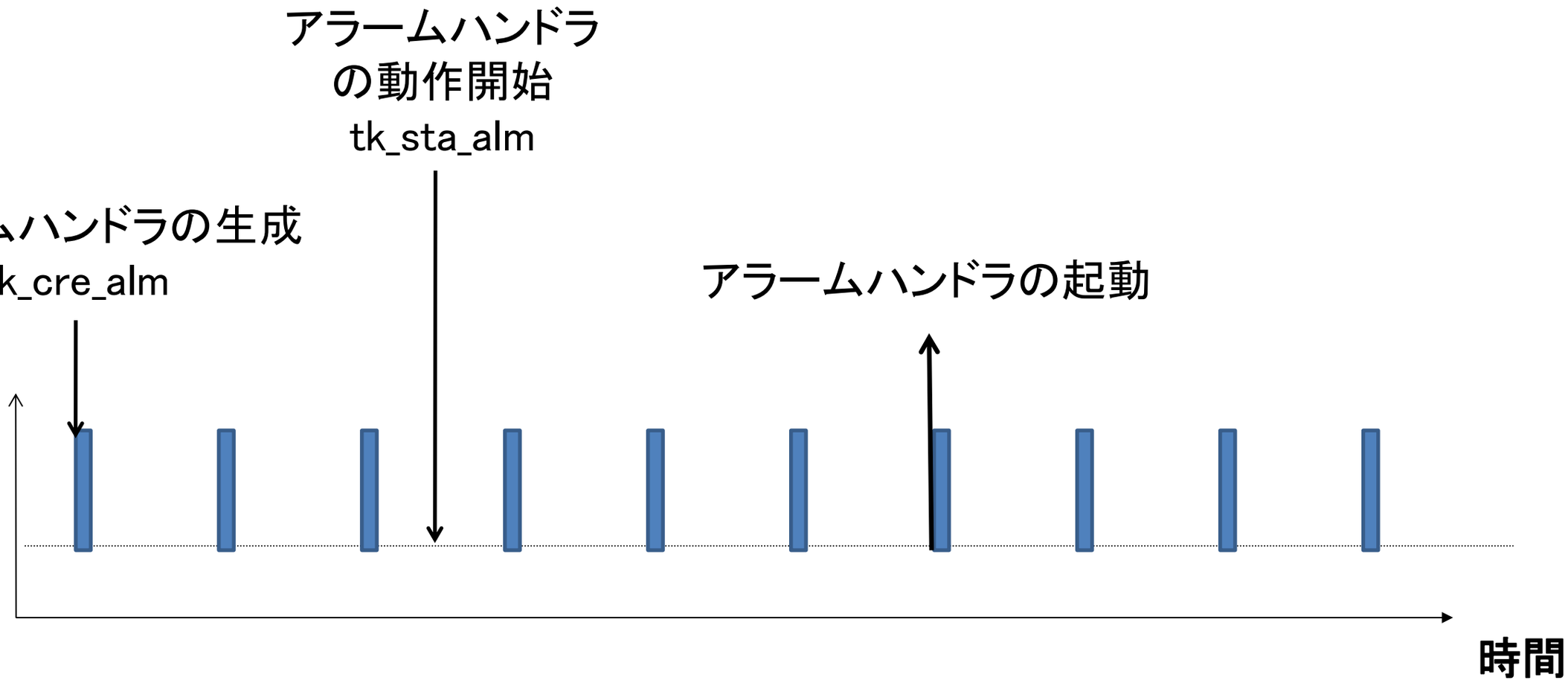
アラームハンドラの動作停止 *Stop Alarm Handler*

```
ER ercd = tk_stp_alm (ID almid);
```

アラームハンドラ状態参照 *Refer Alarm Handler Status*

```
ER ercd = tk_ref_alm (ID almid, T_RALM *pk_ralm);
```

5. 時間管理機能「アラームハンドラ」



【 演習7 周期ハンドラ 】

■ プロジェクトフォルダ
C:¥ut_realos¥utkernel¥7-M¥uT-Kernel_Samples¥cyclic

■ プログラム概要
・ タスクは，lcd1(優先度1)，周期ハンドラ cyc1 から構成される
・ 初期化タスク uinit_task は周期ハンドラとタスクを動的に生成し，タスクを起動させる.
周期ハンドラはlcd1タスクにより動作開始される.
・ lcd1タスクは時間を表示したら，起床待ち状態に遷移する.
また，周期ハンドラ cyc1 は1s毎に起動しており，周期ハンドラの中で時間カウントされている.
時間カウントが終了したらlcd1タスクを起床させる.
起床したlcd1タスクは時間を表示する．以降この処理を繰り返す.

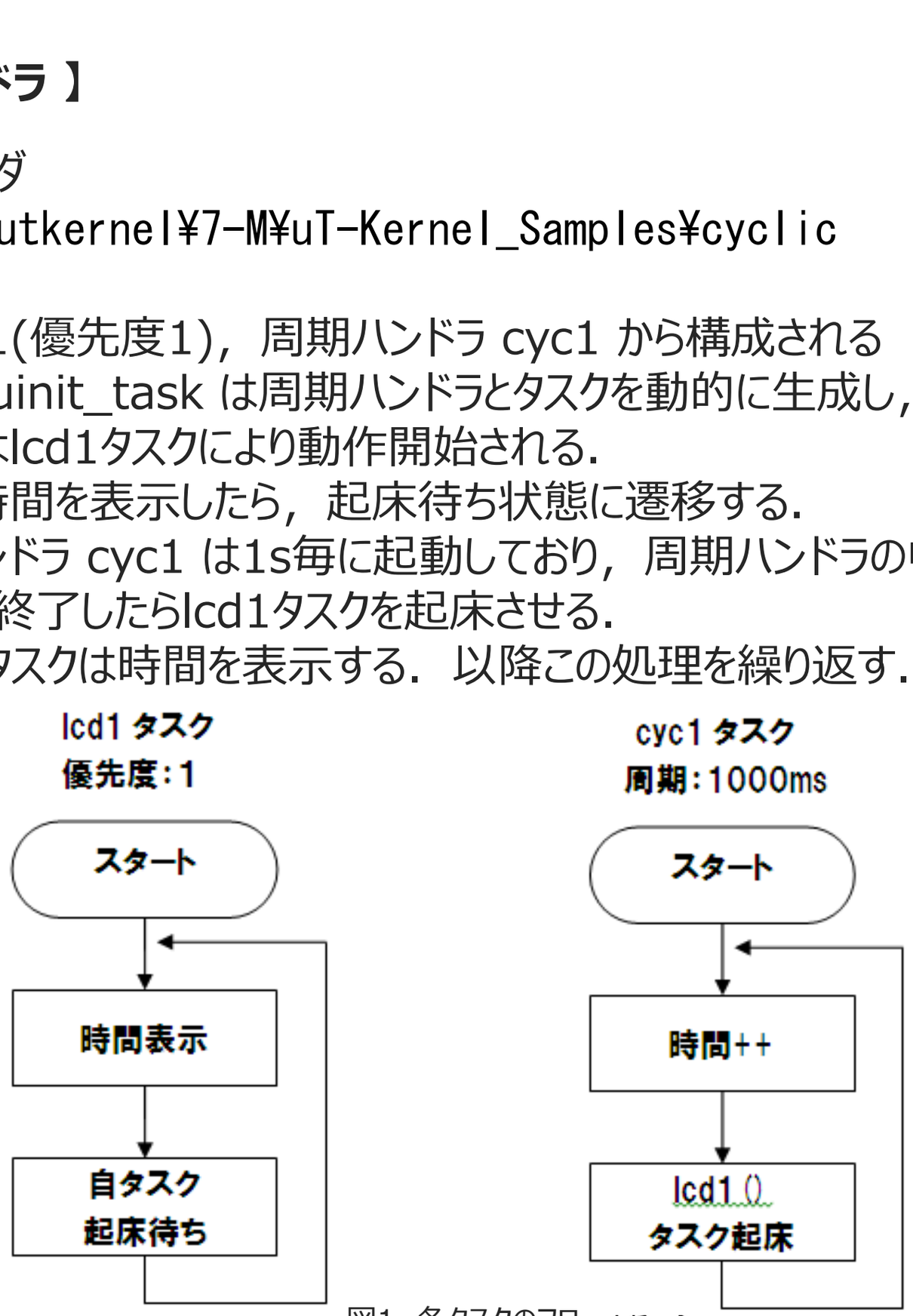


図1．各タスクのフローナヤート

【 演習8 タイムシェアリング 】

■ プロジェクトフォルダ
C:¥ut_realos¥utkernel¥7-M¥uT-Kernel_Samples¥time_share

■ プログラム概要
・タスクは，led1(優先度2)，led2(優先度2) ，led3(優先度2) と
周期ハンドラ cyc1 から構成される
・初期化タスク uinit_task は周期ハンドラとタスクを動的に生成し，各タスクを起動させる.
・各タスクは実行状態になったら該当するLEDを点灯.
また，周期ハンドラ cyc1 は1s毎に起動しており，周期ハンドラの中で各タスクの優先順位を回転させている．これにより全タスクの優先度が同じにもかかわらず，実行中のタスクが順番に切り替わる.

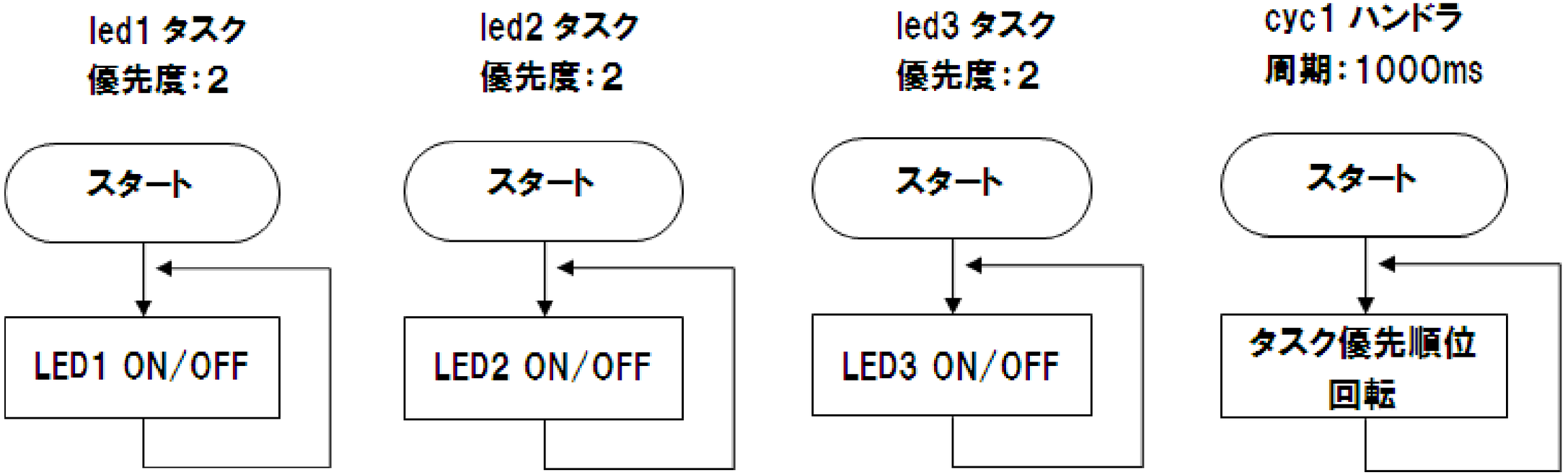


図1．各タスクのフローチャート

【 演習9 総合演習 スロットゲーム 】

■ プロジェクトフォルダ
C:¥ut_realos¥utkernel¥7-M¥uT-Kernel_Samples¥app_slot

■ プログラム概要
・タスクは，「LCD表示タスク(優先度2)」，「スロット回転制御用タスク (優先度1)」，
周期ハンドラ は「スライダーのADC値を利用しスピードをつくるハンドラ」，
「LCD表示を制御する，スロット回転ハンドラ」，「LCDのwaitを制御するハンドラ」
から構成される
・初期化タスク uinit_task は各タスクを起動させる.
・「LCD表示タスク」は「スロット回転ハンドラ」により起こされる.
「スロット回転制御用タスク」は「スピードをつくるハンドラ」により起こされる.
・ADCの値は10ms毎に検出する.
- 回転速度を算出
- LEDの表示を設定(LED8点灯~LED1-8点灯)
- ADCの値に基づいてスロットの回転スピードを設定(100ms~1s)
・PSW1の状態をウォッチ
- PSW1が押されたら，スロットの回転を開始/停止
・スロットが完全に停止したら，勝敗を判定し，結果をLCDに表示

タスク or ハンドラ名	対応する関数名
TSK_LCD_ID	lcd_display()
TSK_SPD_ID	slot_speed()
CYC_ADC_ID	adc_scan()
CYC_TURN_ID	turn_slot()
CYC_LCD_ID	lcd_wait()
割込みハンドラ	psw1()

表1. タスク or ハンドラ名と対応する関数

【 演習9 総合演習 スロットゲーム 】

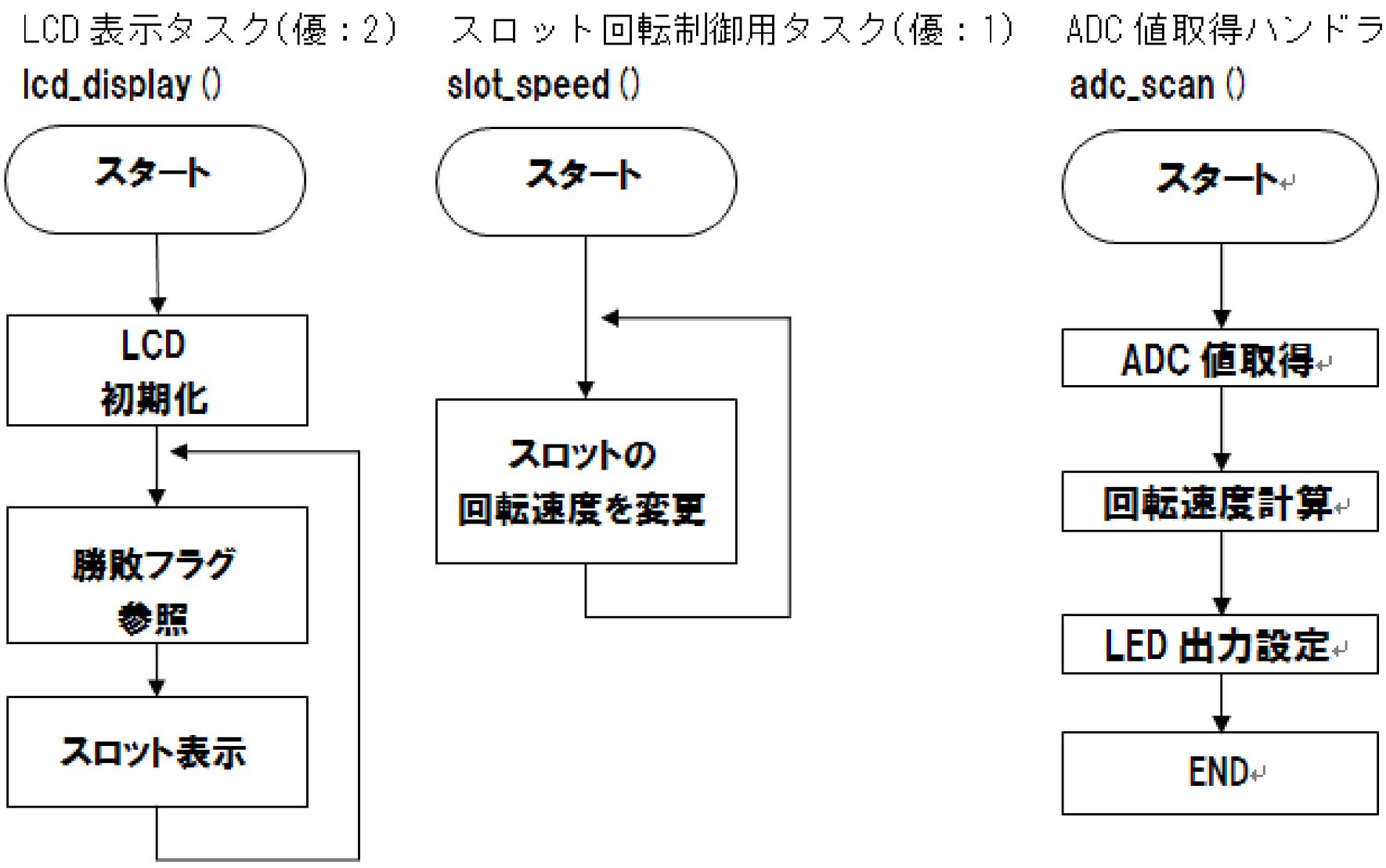


図1. 各タスク or ハンドラのフローチャート

【 演習9 総合演習 スロットゲーム 】

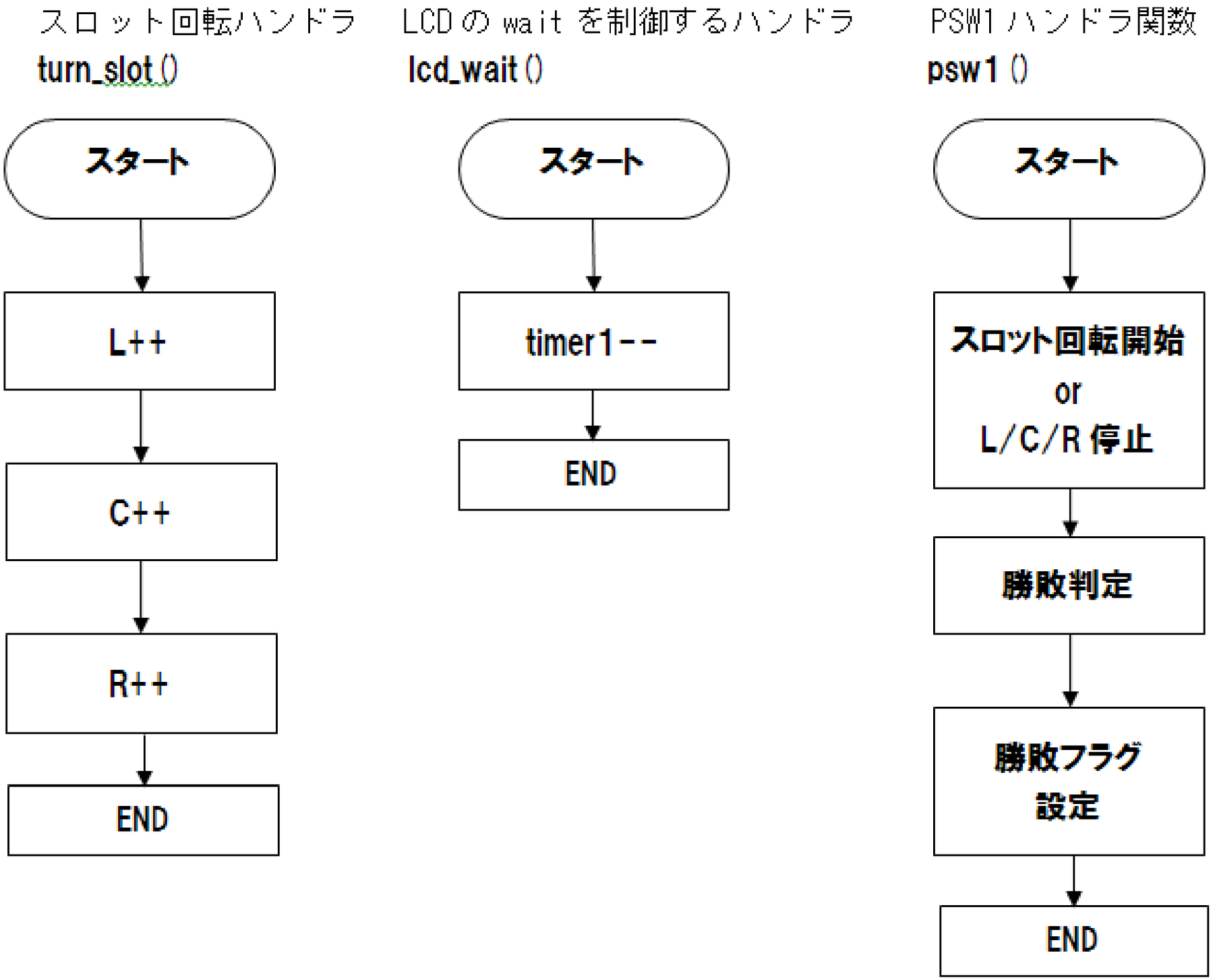


図1. 各タスク or ハンドラのフローチャート

【 演習10 総合演習 音楽プレーヤ 】

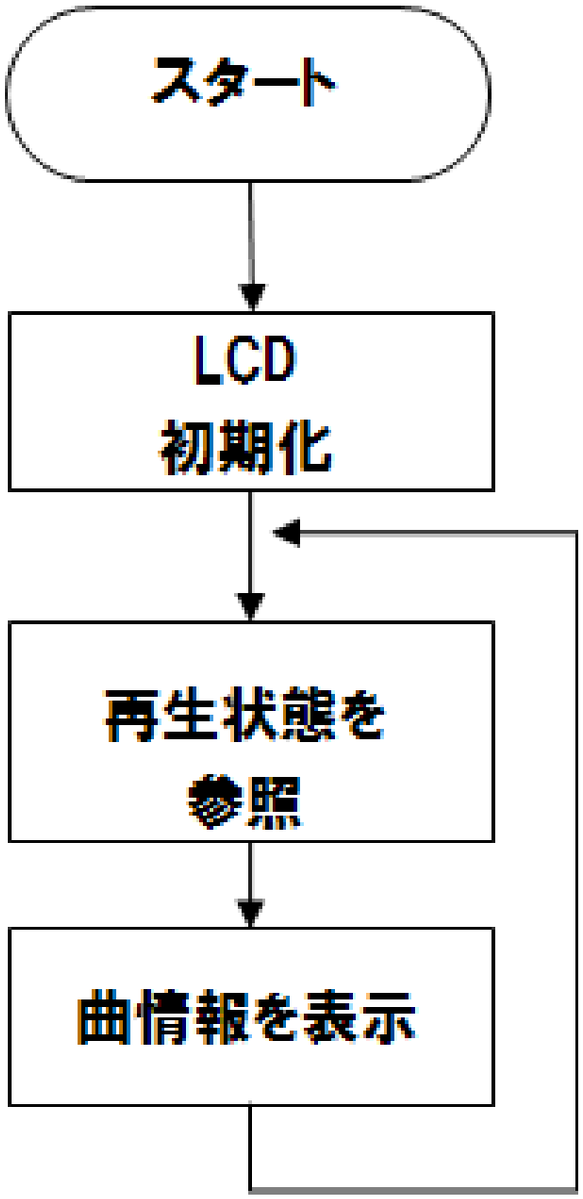
■ プロジェクトフォルダ
C:¥ut_realos¥utkernel¥7-M¥uT-Kernel_Samples¥app_music_player

■ プログラム概要
・ 先頭の曲をLCDに表示
・ ジョイスティック(SW6)で曲を選択
・ PSW1押下で曲を再生
  - 再生時間は00:00からカウントアップ
  - 曲の長さ分再生したら再生停止
  - 再度PSWを押したら一時停止
  - 曲データはLED1～LED8に表示される
・ PSW2押下で再生を停止

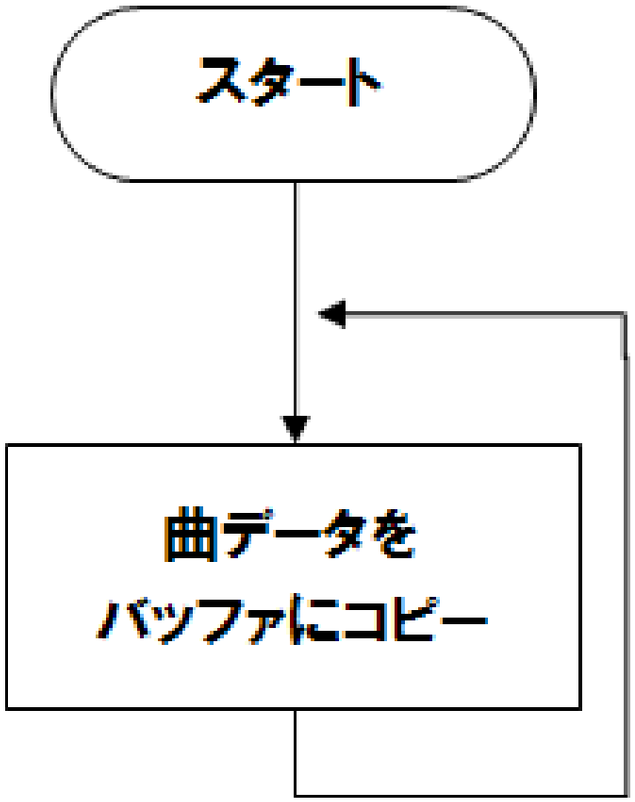
表1. タスク or ハンドラ名と対応する関数

【 演習10 総合演習 音楽プレーヤ 】

LCD 表示関数
lcd_display ()



曲データ読込み関数
data_read ()



入力処理関数
input_scan ()

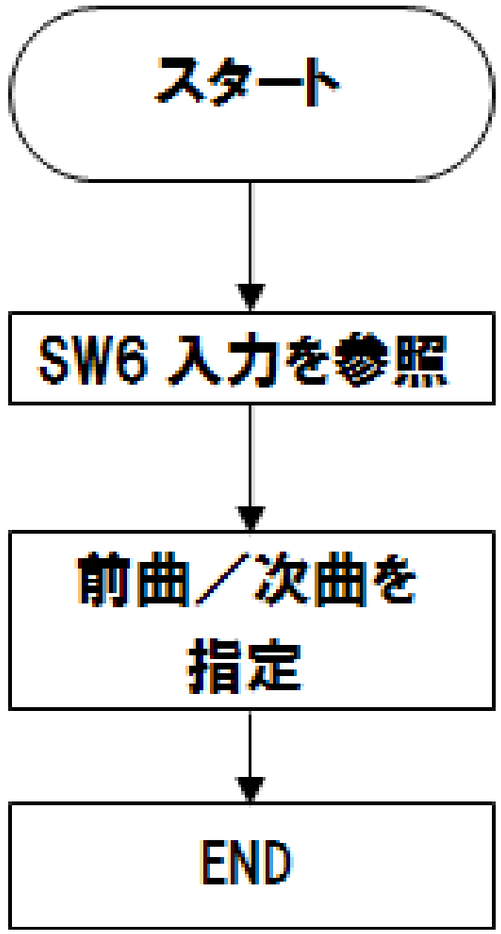
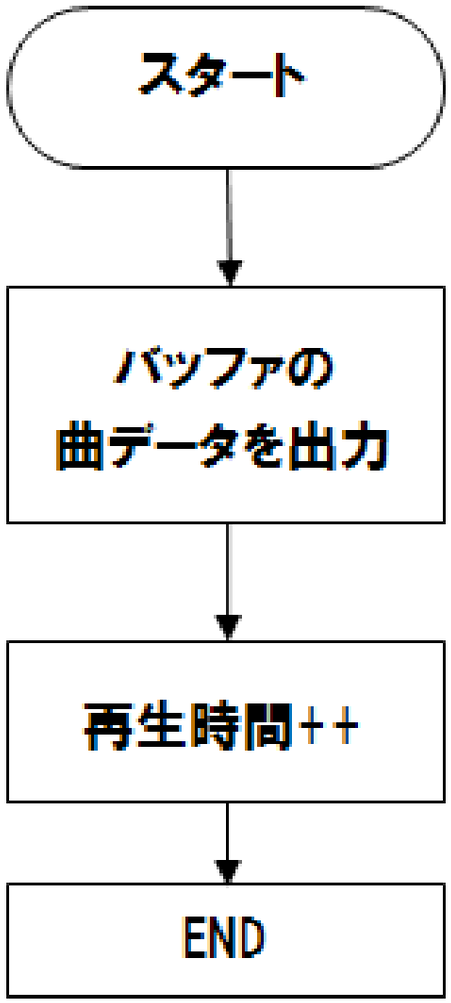


表1. タスク or ハンドラ名と対応する関数

【 演習10 総合演習 音楽プレーヤ 】

曲再生関数

play_count ()



PSW1/PSW2ハンドラ関数

psw ()

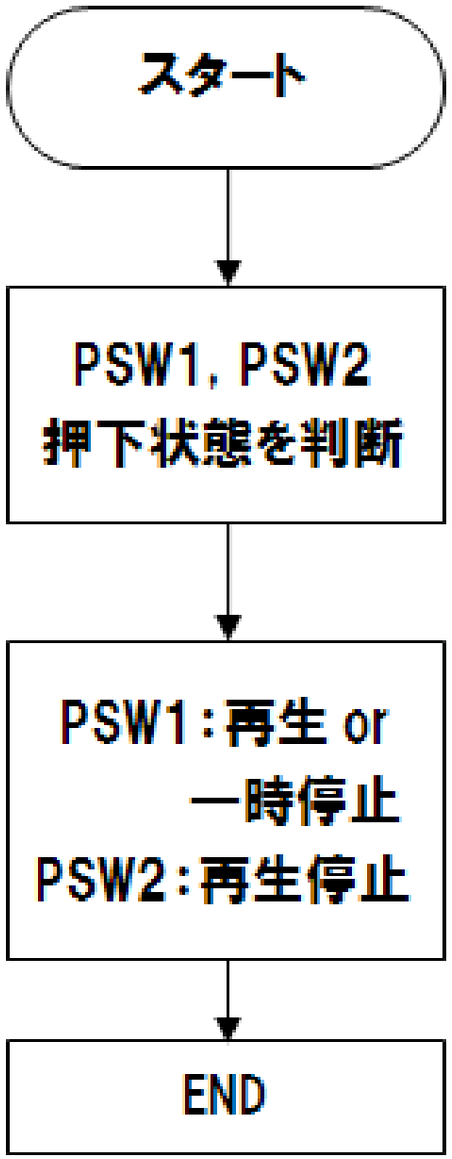
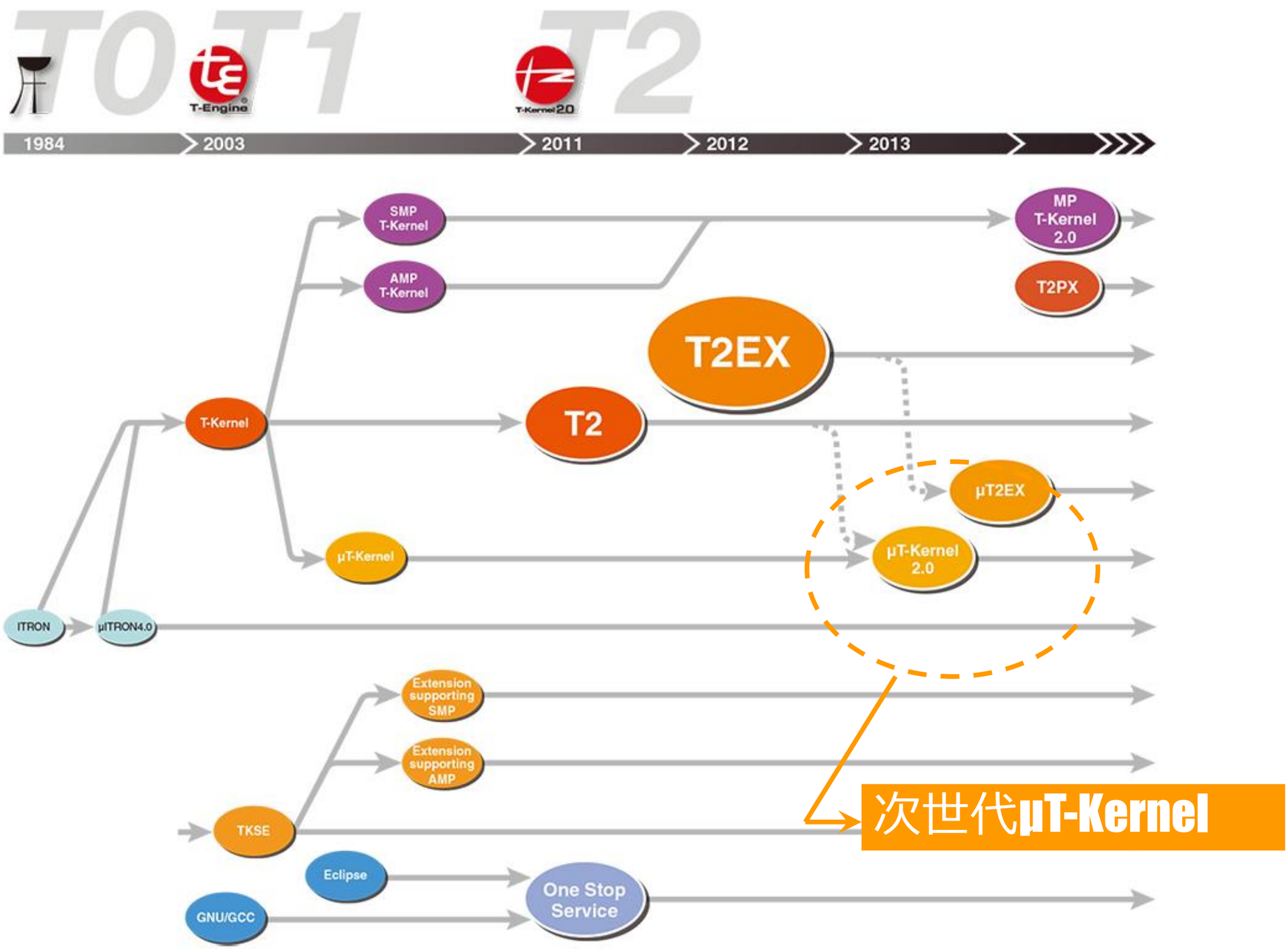


表1. タスク or ハンドラ名と対応する関数

第4章

今後の展開

µT-Kernel2.0



出典：T-Engine フォーラム： http://www.t-engine.org/ja/what-is-t-kernel/t2ex

【実習】μT-Kernel入門(協力：スパンション・イノベイツ)テキスト「プログラミング演習 (1)」

著者　T-Engine Forum