実習:
組込みリアルタイム
プログラミング
(ITRON初級編)

トロンフォーラム 学術・教育WG

1 組込みシステムとマルチタスク・リアルタイム処理

2 トロンと組込みシステム

3 μITRON入門

4 μITRON開発手順

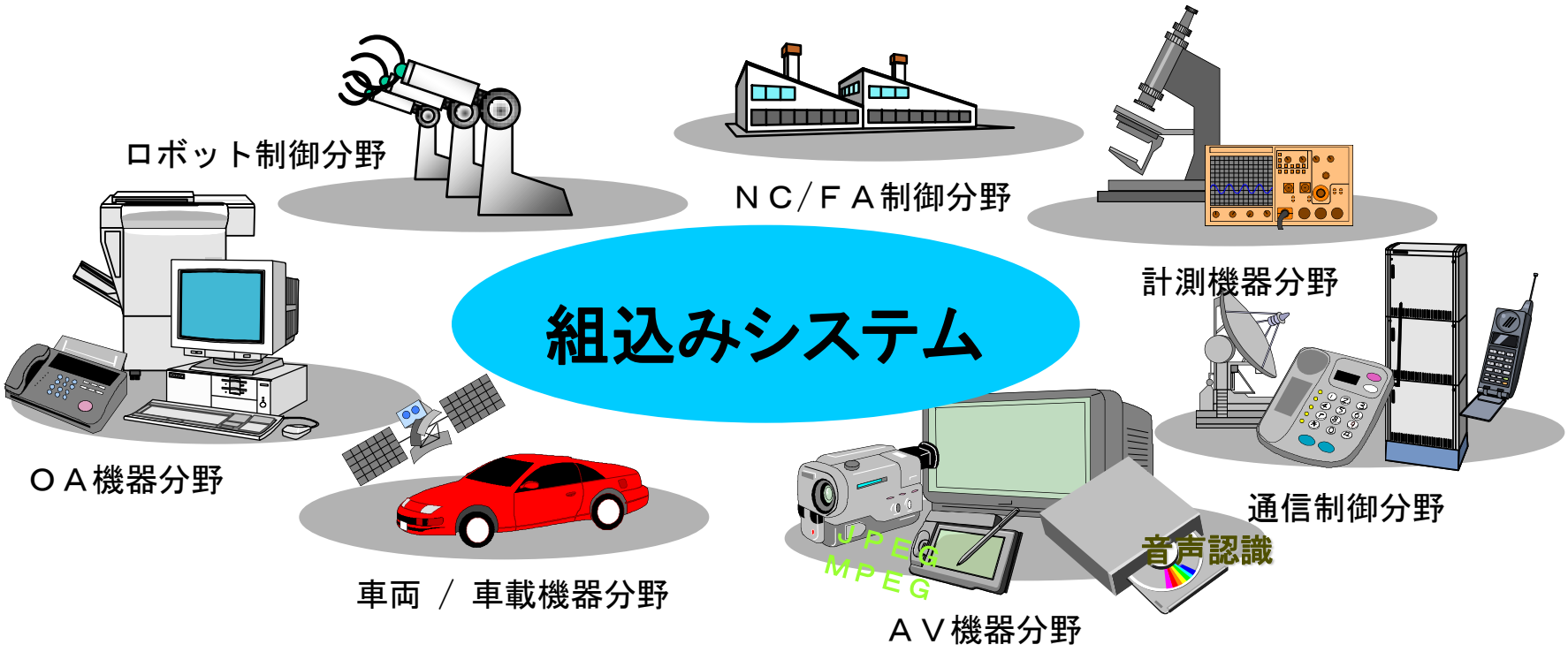
5 μITRONプログラミング

6 μITRONサービスコール

組込みシステムとは

- 組込みシステム=センサやアクチュエータ、他の機械システム等と協調して動作するコンピュータシステム
- (例)
  - 家電製品の制御システム
  - ファックスやコピー機の制御
  - 自動車の制御システム
  - 携帯電話
  - など…

どのようなものが組込みシステムか?



マイコン内蔵炊飯器、洗濯機、コピー機、FAX、
携帯情報端末、電子楽器、自動車、ロボットなど

組込みシステムこそメジャー!

- パソコン・ワークステーション等の出荷数
  - 約1.5億台/年
- 組込みコンピュータの出荷数
  - 約80億台/年
- 米国人が一日に接するマイクロコンピュータの数
  - 100 個[New York Times]

- 組込みコンピュータは今後も増えていく見込み

組込みシステムの特徴(1)

- 計算処理よりも、入出力処理、通信処理が中心
  - イベント処理プログラム
  - 実時間（リアルタイム）処理プログラム
- 必要最小限のハードウェア資源にチューニングする⇒コストを極力下げる

組込みシステムの特徴(2)

- 専用化されたシステム
- 厳しいリソース制約
- 高い信頼性
- システムの改修に多大なコスト
- リアルタイム性

リアルタイム処理とは
(主婦兼母親の朝)

7:00 太郎(5歳)を起こす

7:05 湯が沸く
→紅茶を入れる

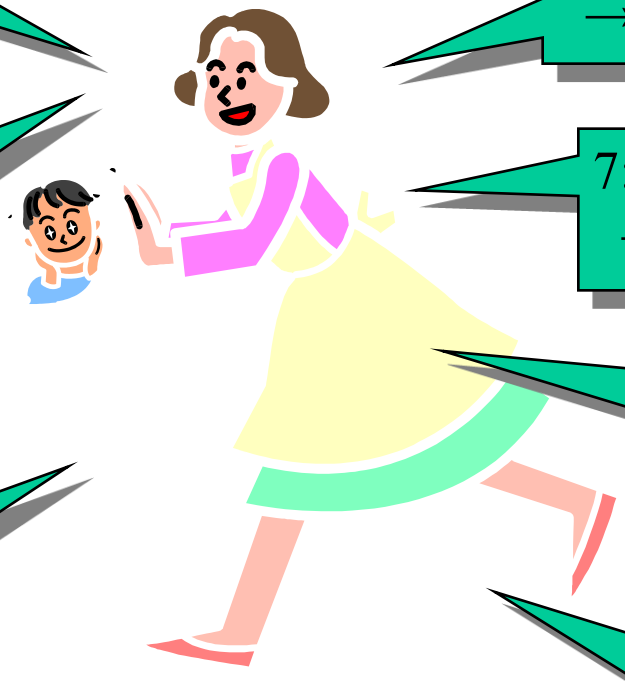
7:15 花子(0歳)おしっこで泣く
→オムツ交換

7:10 トースターが終了
→トーストを皿に

8:10 田中さんから電話
→夕方の買物の約束

8:15 太郎スクールバスに送る

8:28 クリーニング屋
→受け取りと支払い



リアルタイム処理とは
(コピー・プリンタ・ファックス複合機の例)

コピー
原稿セット

コピーボタン
押下

紙づまり

印刷データ
着信

ファックス
着信

リアルタイムシステムとは

- 「リアルタイムシステム」では、計算結果があっていることだけでなく、決められた時間内に計算が終ることも保証しなければならない

(例)

- システムへの要求＝「127＋382の答えを求めなさい。答えは3分後までに出しなさい(12時23分)」
  - (答)509　(12時30分) ⇒リアルタイムシステムでは計算失敗の例となる
  - (答)509　(12時24分) ⇒リアルタイムシステムでも計算成功の例

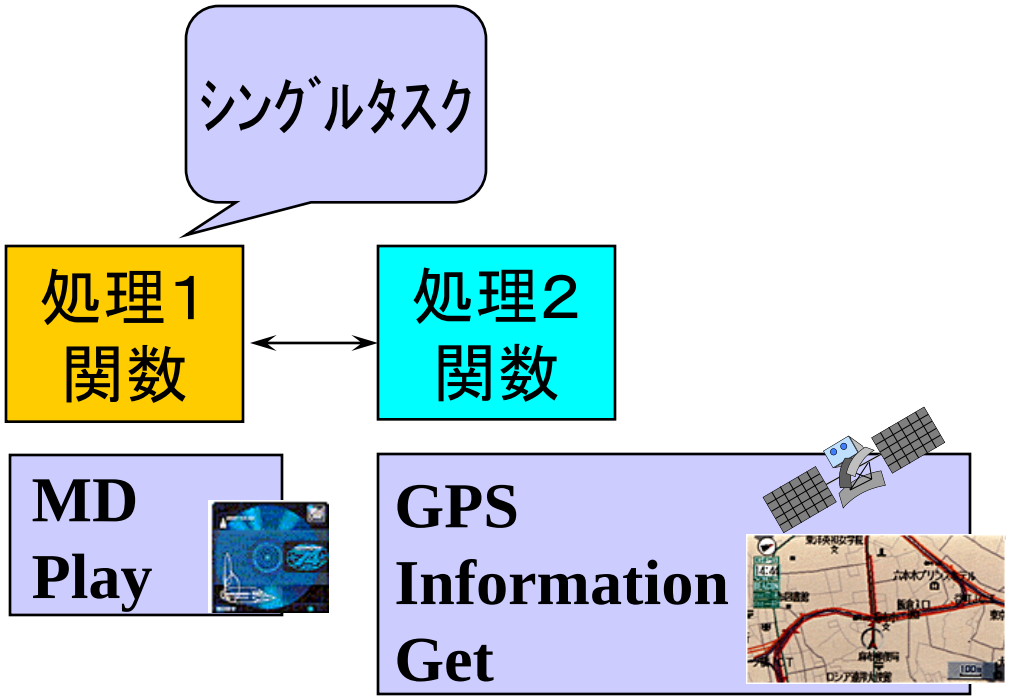
リアルタイムシステムは複雑

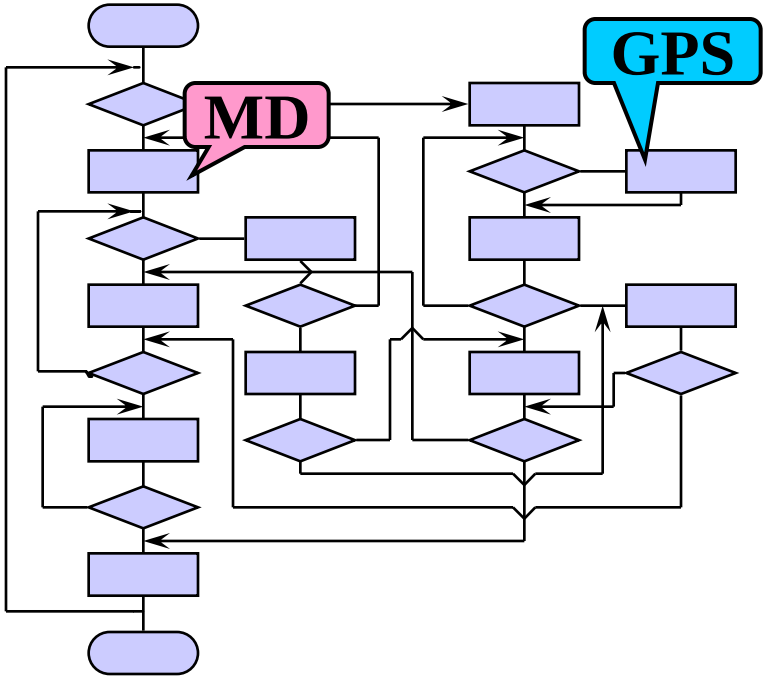
- 実世界とのかかわり
  - ランダムに起きる事象への対応
  - 実時間を扱う必要性
- 複雑な処理が要求されるため、処理するための技術が必要となる
  - 並行処理(タスク、スケジューリング)
  - 同期・通信
  - 実時間処理
  - 記憶管理

なぜリアルタイムOSが必要か？

- すべてユーザプログラムで実現できるか？
  - 実現できても、共通機能として常駐システム化したほうが楽なものも多い
  - 実現できないものもある
- 何らかの汎用的なシステムでサポート
  - リアルタイムOS(Real-time OS : RTOS)

組込み機器制御
(シングルタスク)





プログラムが複雑

→ 規模が大きくなると管理出来ない！

リアルタイムとマルチタスク

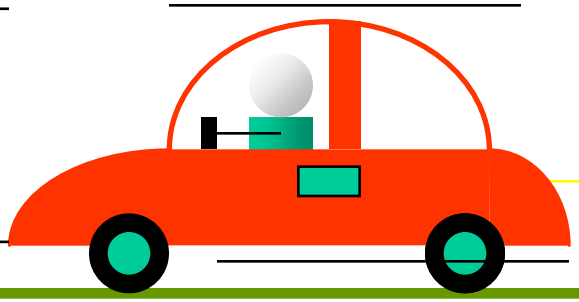
・車の運転の例

スピードが50km/h
以下になったら
アクセルを踏む
<2秒以内>

前方30m以内に
歩行者を発見したら
ブレーキを踏む
<0.1秒以内>

前方30m以内に
赤信号を発見したら
ブレーキを踏む
<1秒以内>

スピードが50km/h
以上になったら
アクセルを離す
<3秒以内>



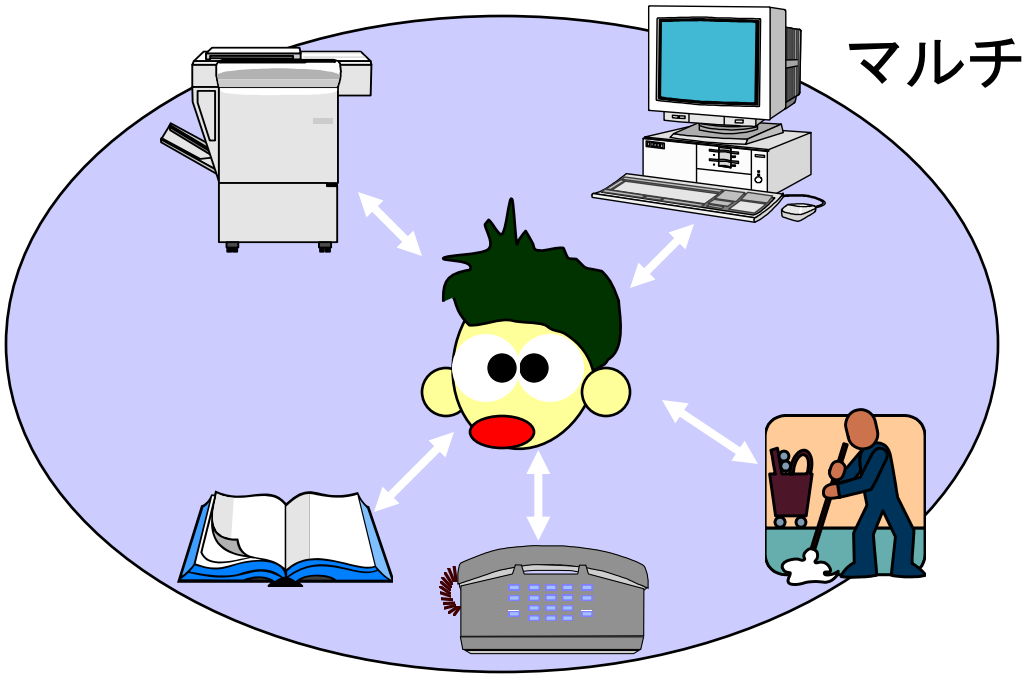
燃料計の針が“E“
を指したらガソリン
スタンドに入る
<15分以内>

制限時間内に**複数**の処理を行う　という

リアルタイムシステム　は、**マルチタスクシステム**　になる場合が多い！

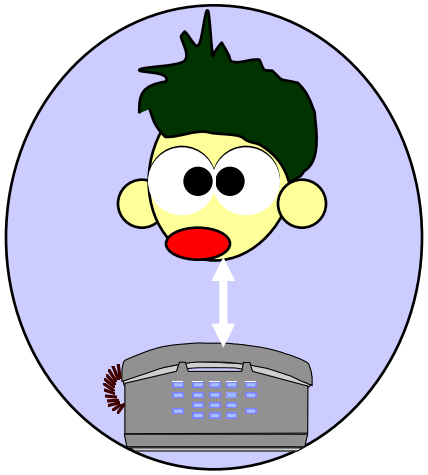
マルチタスクとは

- 複数の処理を同時に実行すること=マルチタスク処理



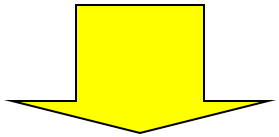
マルチタスク

シングルタスク



タスクスケジューリング

- 同時に複数の処理をさせる(マルチタスク)
- しかしCPUは一つしかない…
- CPUを使う時間帯を複数のタスクに割り振る



- タスクスケジューリング

RTOS導入のメリット

- リアルタイム処理の基本処理を扱ってくれる=プログラム作成が容易に
  - 並行処理・スケジューリングのサポート
    - タスク分割設計が容易に、効率的なI/Oも簡単に実現
  - 同期処理のサポート
    - タスク間の通信・同期の実現が容易に
  - 記憶管理のサポート
    - 記憶管理の細部に関らないでもすむ。
- ハードウェアの相違点をRTOSが吸収してくれるので、プログラムの再利用性が向上
- モジュール化がしやすい、関連製品が利用可能などにより、保守性・拡張性が向上

1 組込みシステムとマルチタスク･リアルタイム処理

2 トロンと組込みシステム

3 μITRON入門

4 μITRON開発手順

5 μITRONプログラミング

6 μITRONサービスコール

トロンプロジェクトとITRON

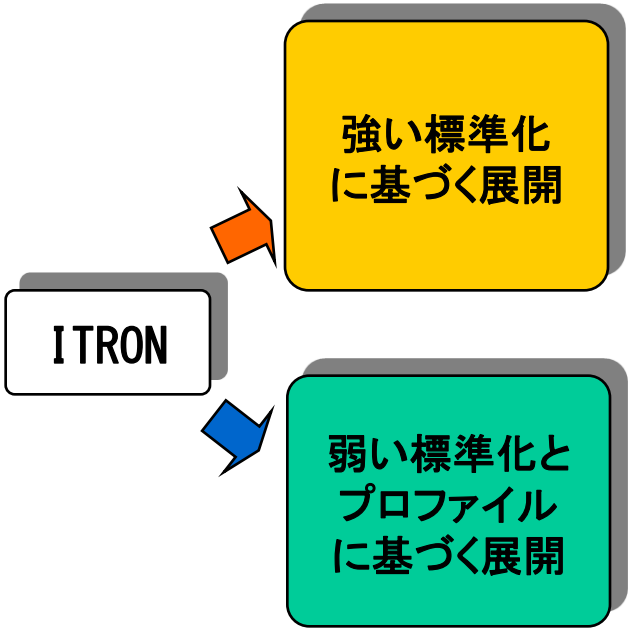
- TRON Project since 1984
  - トロンプロジェクトはUbiquitous Computing研究開発の先駆け
- トロン(TRON)
  - The Real-time Operating system Nucleus
- 組込みシステム向けに、ITRON(Industrial TRON) Specification を公開(1989)
  - 現在のバージョンはμITRON 4.0仕様

トロンの有効性

- トロン(The Real-time Operating system Nucleus)の特長
  - 仕様が公開されている(オープン)
  - 実装が自由(ライセンス料がかからない)
  - 国内各社で利用されている(ITRON/T-Kernel)
- なぜ各社で利用されているのか？
  - ソフトウェアの流通性・再利用性の確保
  - 企業内のソフトウェア開発の標準化
  - 技術者教育の標準化

ITRONとT-Kernel

ITRONの成果を生かしつつ、標準化の範囲を拡大し、高度な技術を取り入れることで、短期間で高度な組込みシステムを作るためのソリューションの整備



T-Kernel

ユビキタスコンピューティング環境構築を目指して、オープンなリアルタイム標準開発環境を整備するプロジェクト。 ハードウェア、OS、ミドルウェア、開発環境について、強い標準化が進められている。

ITRON4.0仕様

- 同一プロファイル規定のOS間で互換性を保証
  （スタンダードプロファイル、自動車プロファイル）
- 保護機能拡張(ITRON4.0/PX仕様)によるMMUサポート(メモリ保護を主目的としている)

ITRONの成果をベースとして、互換性や厳密性の向上を図り、生産性向上や再利用性、移植性を高めることを目的に仕様を整理

μITRON 4.0とT-Kernelの機能比較(概略)

機　能	μITRON 4.0	T-Kernel	機能差
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	○	△	TKはﾃﾞｰﾀｷｭｰ無
拡張同期・通信機能	○	○	
メモリ・プール機能	○	○	
時間管理機能	○	△	TKはｵｰﾊﾞﾗﾝH無
システム状態管理機能	○	○	
割込み管理機能	○	○	
サービスコール管理機能	○	×	TKはｻﾌﾞｼｽﾃﾑが同等
システム構成管理機能	○	○	
サブシステム管理機能	×	○	
システム・メモリ管理機能	×	○	
アドレス空間管理機能	×	○	
デバイス管理機能	×	○	
I/Oポート・アクセスサポート機能	×	○	
省電力機能	×	○	

参考:その他の組込み向けOS

- VxWORKS
  - ソフトウェアベンダ系
- 組込みLinux
  - UnixクローンOS Linuxの組込み版
- AUTOSAR
  - 車載向け、欧州

1 組込みシステムとマルチタスク･リアルタイム処理

2 トロンと組込みシステム

3 μITRON入門

4 μITRON開発手順

5 μITRONプログラミング

6 μITRONサービスコール

μITRONが提供する機能

- 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
- 拡張同期･通信機能
- メモリ･プール管理機能
- 時間管理機能
- システム状態管理機能
- 割込み管理機能
- その他の機能
  - サービスコール管理
  - システム構成管理

タスクについての規定

- タスクは、RTOS上で最小実行単位となるサブルーチン
- 各タスクには優先度を指定

- タスクには以下の状態があり、順次切り替えて並行動作を行う(タスクスケジューリング)
  - 実行可能状態、実行状態、待ち状態、強制待ち状態、二重待ち状態、休止状態、未登録状態

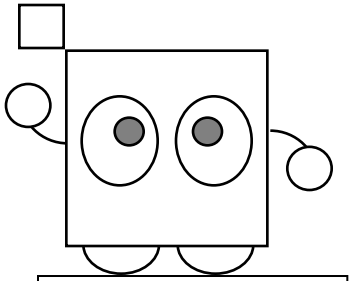
タスクの状態(1)

- 実行状態（RUNNING)
  - 現在タスクが実行中の状態
  - 実行可能状態のタスクの中で最も優先順位の高いタスクの状態
- 実行可能状態（READY)
  - sta_tsk等により実行可能になった状態
- 待ち状態（WAITING）
  - 待ち解除の条件が満たされるのを待っている状態
  - 条件が満たされると実行可能状態に移行
- 強制待ち状態（SUSPENDED）
  - 他タスクから強制的に中断された状態
  - スケジューリングの対象から強制的にはずされたタスクの状態

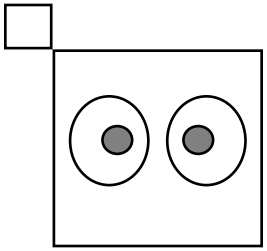
タスクの状態(2)

- 二重待ち状態（WAITING-SUSPENDED）
  - 待ち状態と強制待ち状態の2つの待ち状態が重なった状態
  - 待ち状態のタスクに対して、sus_tskが発行された状態
- 休止状態（DORMANT)
  - タスクが起動されるのを待っている状態
  - タスクの起動前、および タスク終了後の状態
- 未登録状態（NON-EXISTENT)
  - タスクが登録されていない状態
  - システムから削除された状態

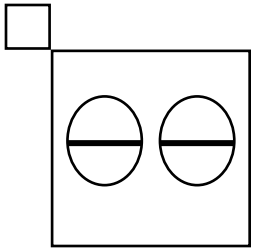
タスクの状態(アイコン)



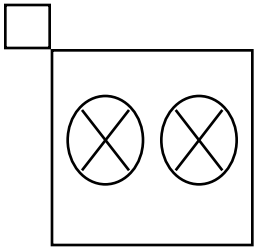
実行状態
RUNNING



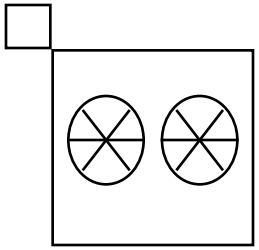
実行可能状態
READY



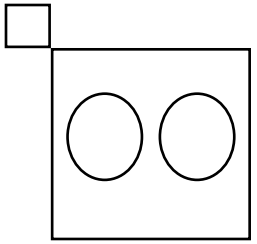
待ち状態
WAITING



強制待ち状態
SUSPENDED

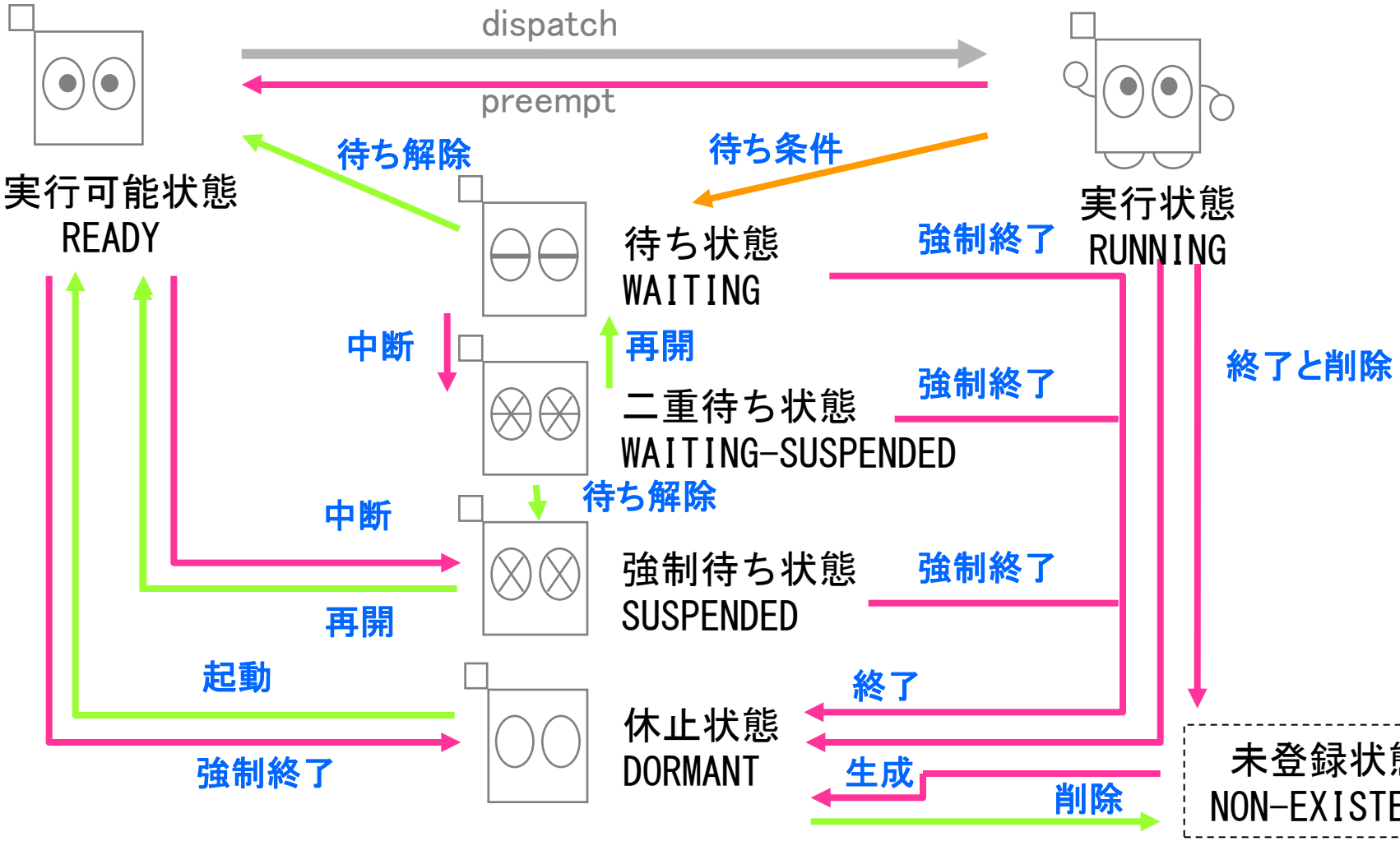


二重待ち状態
WAITING-SUSPENDED

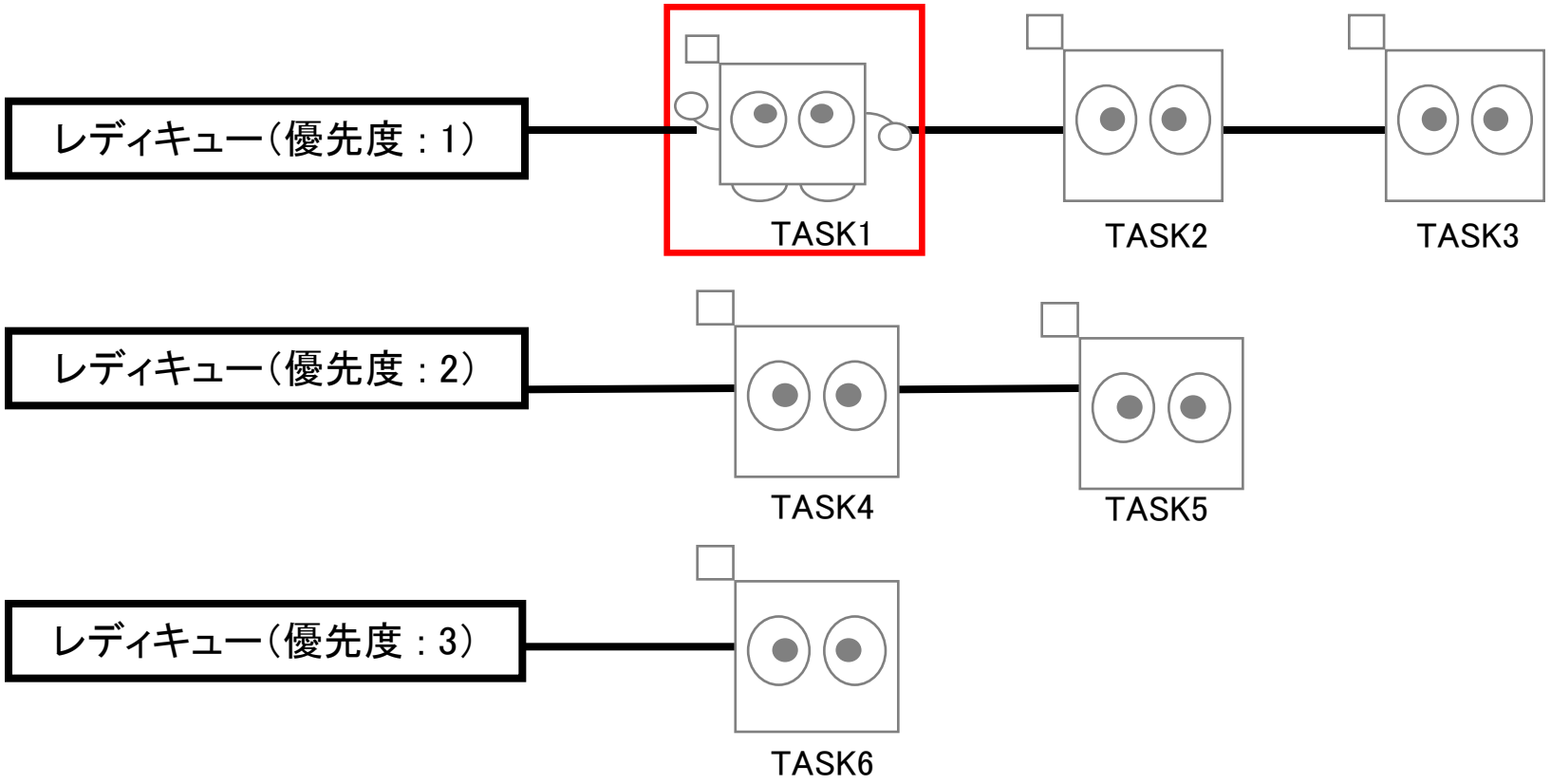


休止状態
DORMANT

タスク遷移の様子


dispatch
preempt
待ち解除
待ち条件
実行可能状態
READY
待ち状態
WAITING
実行状態
RUNNING
強制終了
中断
再開
終了と削除
二重待ち状態
WAITING-SUSPENDED
強制終了
待ち解除
中断
強制待ち状態
SUSPENDED
強制終了
再開
起動
終了
休止状態
DORMANT
生成
削除
強制終了
未登録状態
NON-EXISTENT

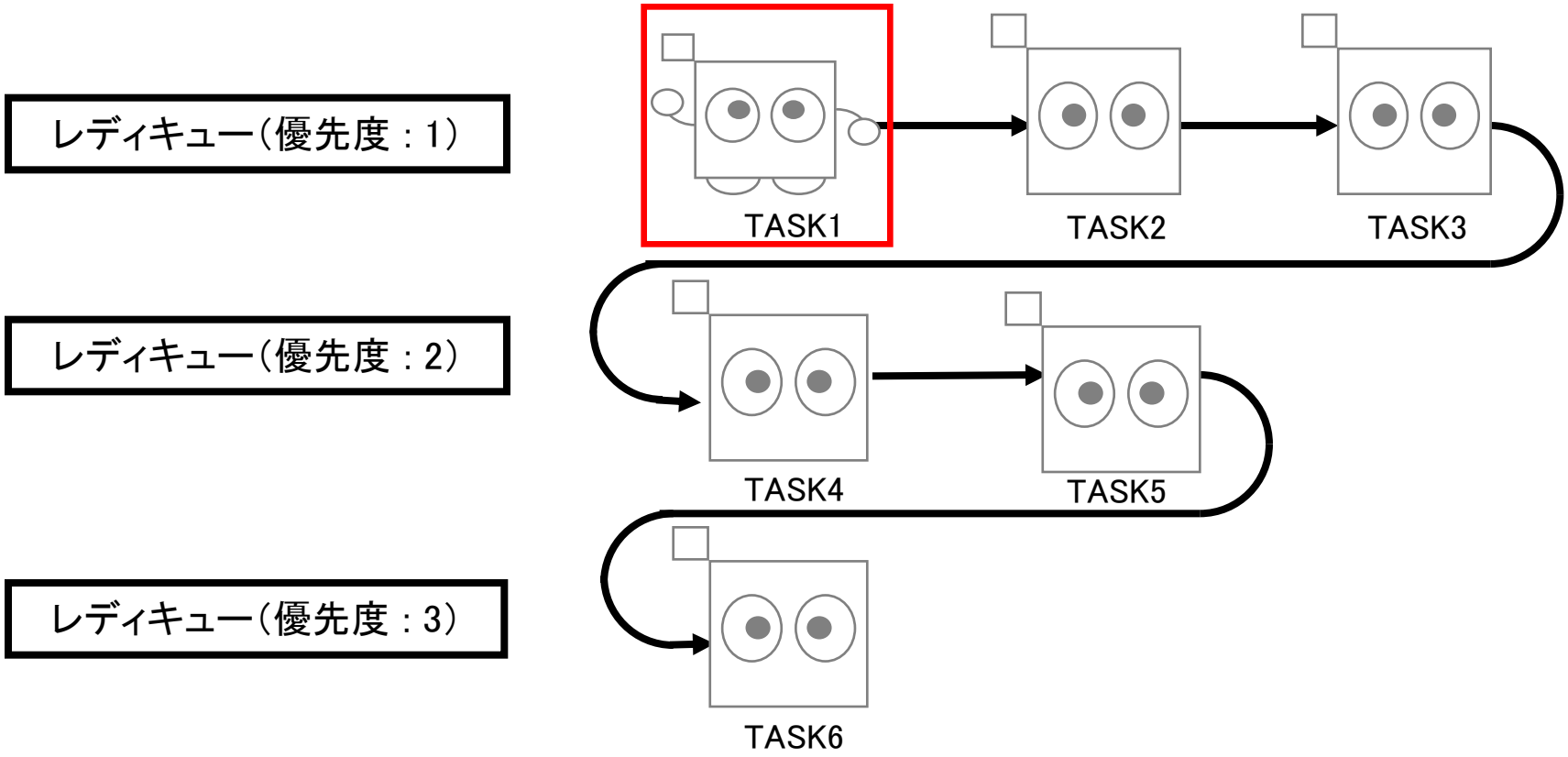
レディーキュー



・実行可能状態にあるタスクのつながる行列

・最も優先順位の高いタスクが実行状態になる

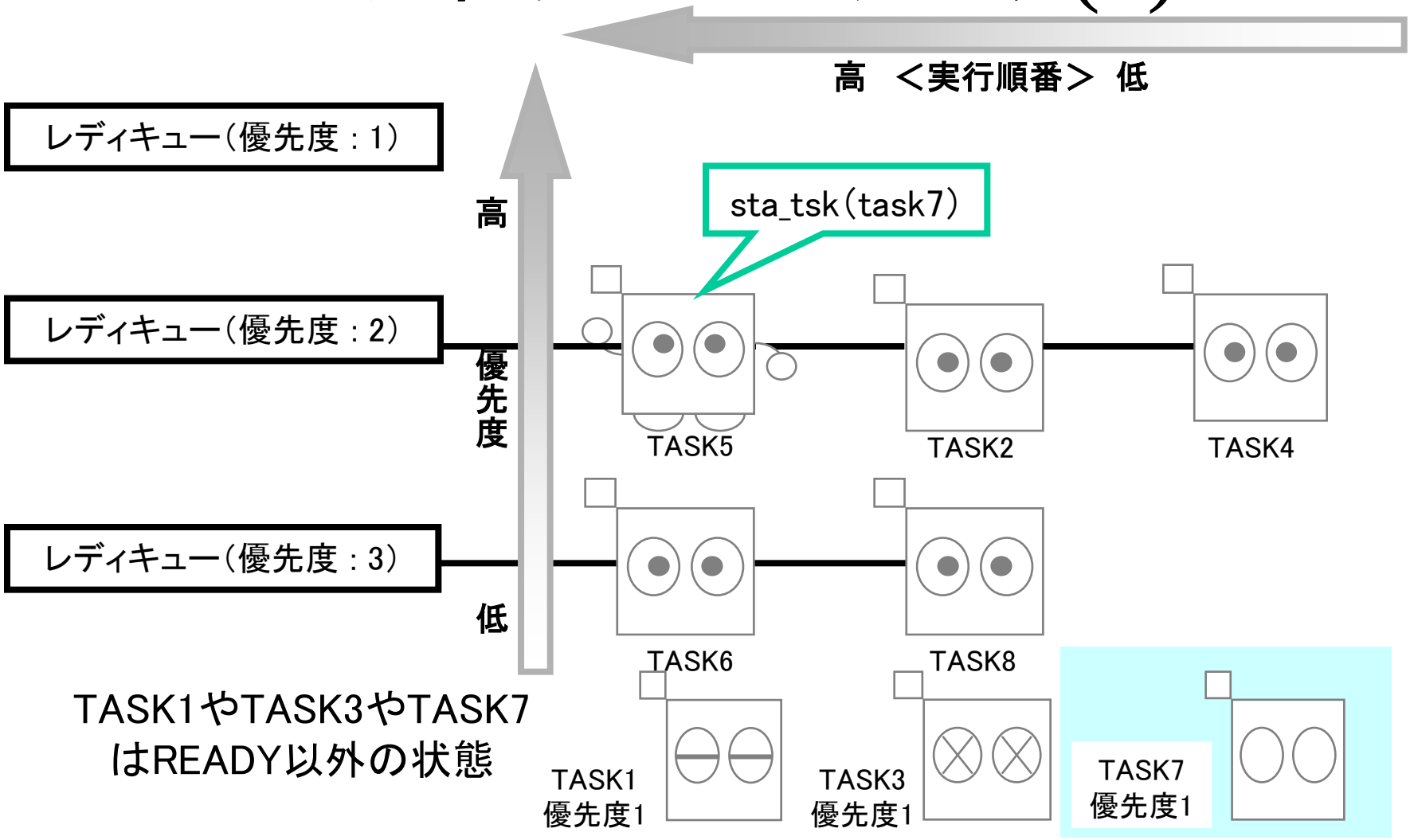
タスクの実行順序



・優先順（優先度の数字が小さい順）

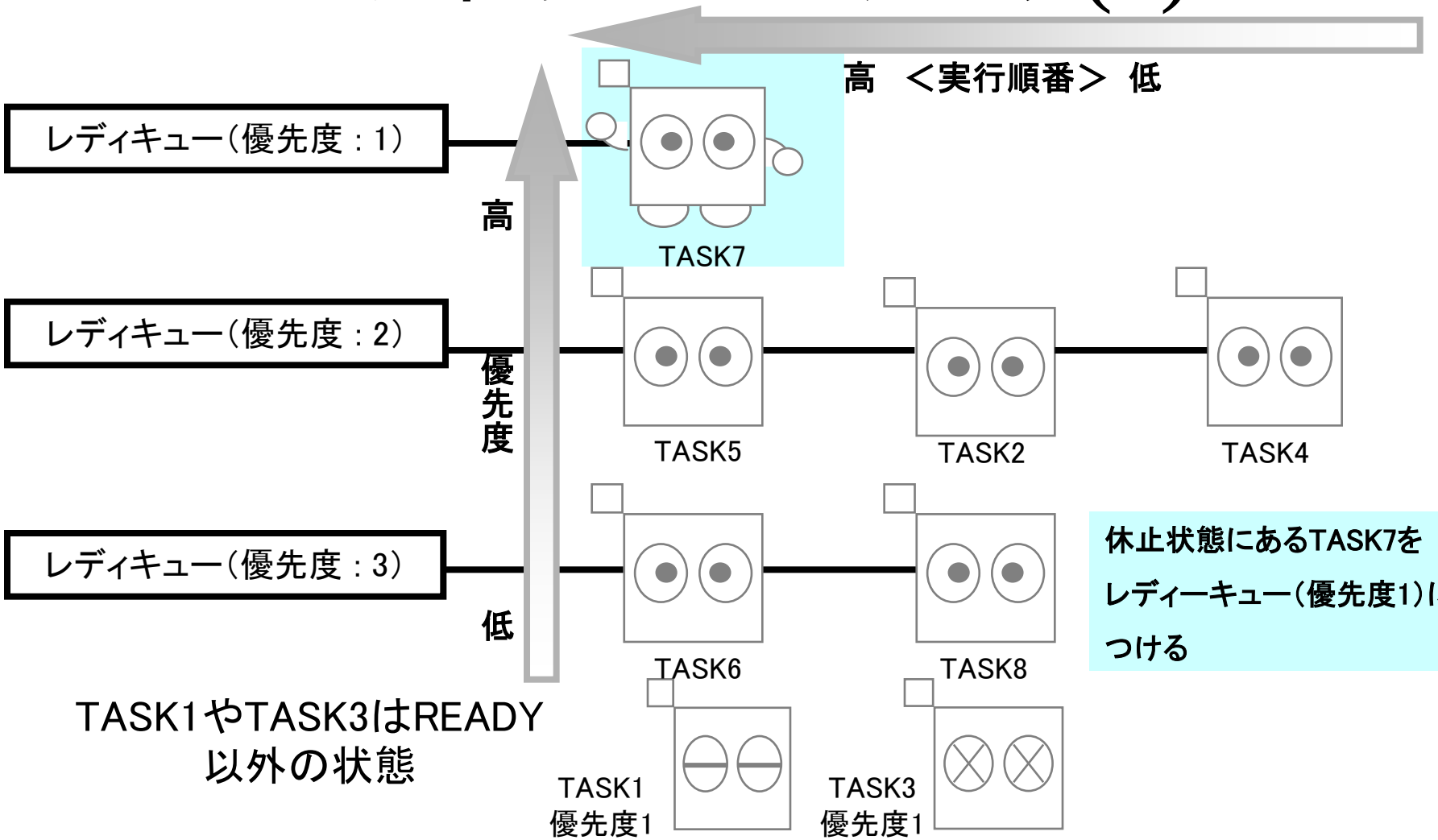
・同じ優先度なら先にレディーキューに並んだ順（FIFO順）

レディキューのタスク(1)



TASK1やTASK3やTASK7はREADY以外の状態

レディキューのタスク(2)



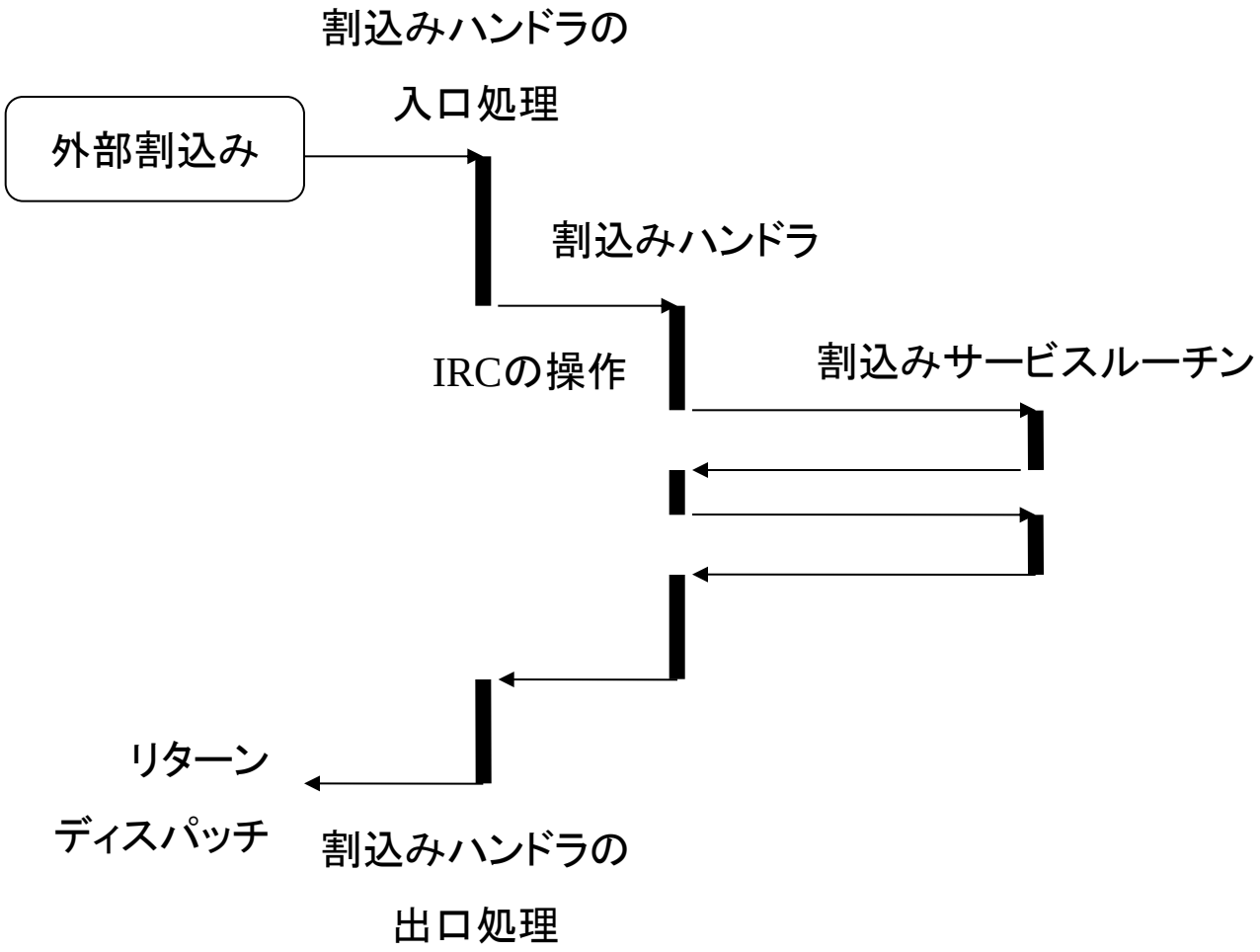
タスク動作以外の規定

- 用語、ID番号、優先度、エラーコード等
- ITRON共通規定として用意されている
  - ソフトウェア部品仕様にも共通に適用可能
  - 規定を理解しておくことで、仕様全体も理解しやすい(プログラム可読性も向上)
- μITRON3.0/4.0/T-Kernelにおいて、基本的な概念や扱いは共通

割込み処理モデル

- 外部割込みによって起動される処理は、割込みハンドラと割込みサービスルーチン
  - 割込みハンドラはプロセッサの機能のみに依存して起動される
    - 割込みコントローラ(IRC ; Interrupt Request Controller)の操作が必要になる場合がある
  - 割込みサービスルーチンは割込みハンドラから起動される
    - IRCに依存しない
- いずれかの機能が提供されていれば良い

割込み処理モデル


割込みハンドラの
入口処理
外部割込み
割込みハンドラ
IRCの操作
割込みサービスルーチン
リターン
ディスパッチ
割込みハンドラの
出口処理

ITRON共通規定 概要(1)

- 用語の規定
  - サービスコール、コールバック、静的API、オブジェクト、他
- オブジェクトのID番号
  - オブジェクト毎に、数値(番号)で識別
  - 1から連続した正の値
- 優先度の数値
  - 1以上の正の値
  - 値が小さいほど優先度が高い

ITRON共通規定 概要(2)

- エラーコード
  - 正常終了はE_OK(=0)または正の値
  - 負の値はエラー
  - E_xxxでエラーコードを定義
- システムコンフィギュレーション
  - 静的APIによる、システムの初期状態の設定

ITRON共通規定 概要(3)

- APIの名称に関する原則
  - xxxで操作方法、yyyで操作対象
  - xxx_yyy()の形が基本
  - ハンドラから呼び出せるサービスコールには、識別名の前に“**i**”を付加
  - 独自に定義したサービスコールには、識別名の前に“**v**”を付加することを推奨

1 組込みシステムとマルチタスク･リアルタイム処理

2 トロンと組込みシステム

3 μITRON入門

4 μITRON開発手順

5 μITRONプログラミング

6 μITRONサービスコール

コンフィギュレーション

- RTOSの動作環境をコンフィグレーションファイルに記述
- 静的APIによってオブジェクトを生成
  - ID番号の指定
  - 最大オブジェクト数の指定など
- 割込みハンドラを指定
- コンフィギュレータを使用してヘッダファイルとカーネル情報ファイルの出力

•コンフィギュレーション方法はOSによって方法などが異なる場合があります

静的APIの例

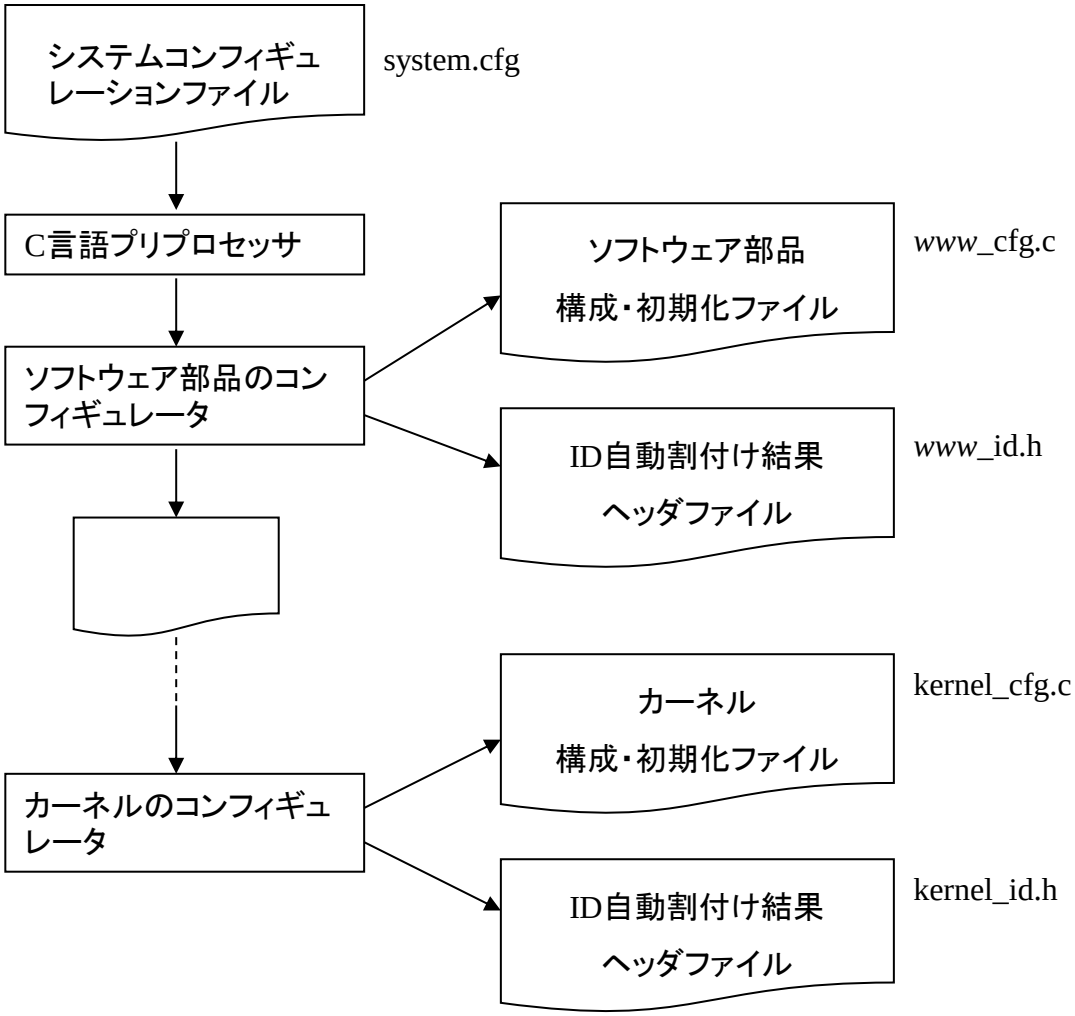
CRE_SEM

機能	セマフォの生成
API	CRE_SEM(ID semid,{ATR sematr,UNIT isemcnt,UINT maxsem});
備考	semid　セマフォのID番号 sematr　セマフォの属性（FIFO待ち or 優先度待ち） isemcnt セマフォの初期値 maxsem　セマフォの最大値

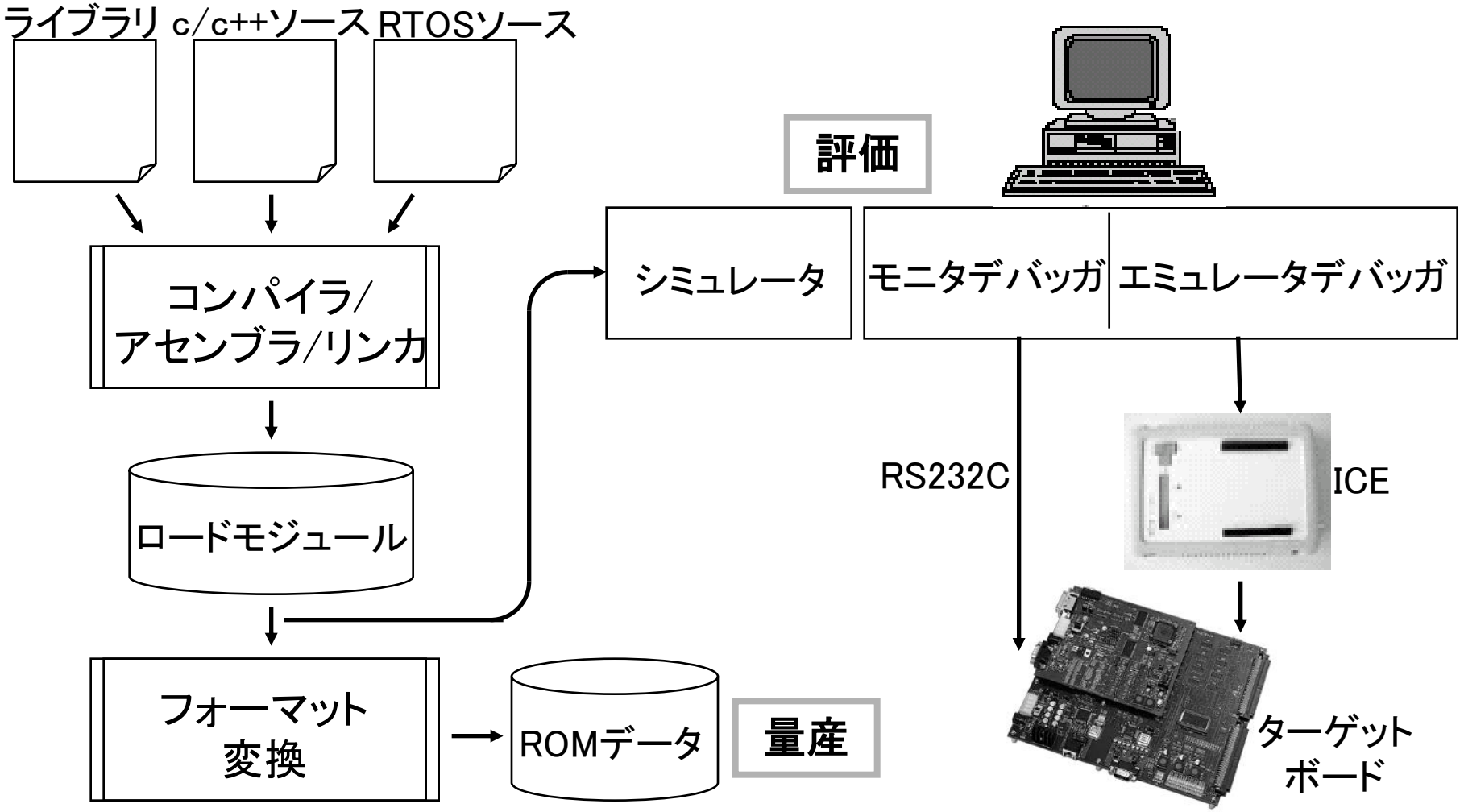
コンフィギュレーションファイルの例

```
/*************************************************/
/* Sample configuration file for uITRON 4.0    */
/*************************************************/

INCLUDE("demo.h");

#define STACK_SIZE 2048
CRE_TSK(TASK1, {TA_HLNG|TA_ACT, TASK1, task1, 15, STACK_SIZE, NULL});
CRE_TSK(TASK2, {TA_HLNG|TA_ACT, TASK2, task2, 15, STACK_SIZE, NULL});

CRE_FLG(ID_FLG1, {TA_TFIFO|TA_CLR|TA_WMUL, 0});
CRE_SEM(SEM1, {TA_TFIFO, 0, 10});
CRE_DTQ(DTQ1, {TA_TPRI, 10, NULL});

CRE_CYC(ID_CYC1, {TA_HLNG|TA_STA, ID_CYC1, cyc_func, 4000, 2000});
```

コンフィギュレーションの流れ



開発環境の例


ライブラリ
c/c++ソース
RTOSソース
コンパイラ/
アセンブラ/リンカ
ロードモジュール
フォーマット
変換
ROMデータ
量産
評価
シミュレータ
モニタデバッガ
エミュレータデバッガ
RS232C
ICE
ターゲット
ボード

1 組込みシステムとマルチタスク･リアルタイム処理

2 トロンと組込みシステム

3 μITRON入門

4 μITRON開発手順

5 μITRONプログラミング

6 μITRONサービスコール

一般的なC言語プログラミングと違う点(1)

- μITRONでは、タスクはmain()から起動するとは限らない
  - 組込み機器では、resetから始まる
  - PCでもresetから動作しているが、使用者には見えないようになっている
- イベントが発生しないと、タスクは切替らない
  - 割込みやサービスコールの呼出し等のイベントが発生し、動作中のタスクが待ち状態に入った時に、他のタスクに切替る

一般的なC言語プログラミングと違う点(2)

- イベントが発生し、タスクが切り替え可能状態であれば、優先度が高いタスクに切替る(プリエンプション)
  - プリエンプション動作により、高い優先度のタスクが動作するので、高速な応答を行う必要があるタスクには、高い優先度を設定できる
  - 意識的に切り替え(ディスパッチ)を禁止することも可能
- リアルタイム性の確保はアプリケーションの作り方も重要

イベントドリブン型プログラミング

- リアルタイムOSでは、何か要求(イベント)がきたら動作する、というイベントドリブン型プログラミングが基本
  - 受動型プログラムとも呼ばれる
  - 発生したイベントに対する高速応答が可能
- 自ら動作を行う、能動的なプログラミングも可能
  - すべて能動的な動作では、リアルタイム性を確保するのは難しい場合が多い
- どのような形式にするかはアプリケーションによる

1 組込みシステムとマルチタスク･リアルタイム処理

2 トロンと組込みシステム

3 μITRON入門

4 μITRON開発手順

5 μITRONプログラミング

6 μITRONサービスコール

代表的なサービスコール

- 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
  - データキュー
  - メールボックス
- 時間管理機能

タスク管理機能

- タスク管理機能は、タスクの状態を直接的に操作/参照するための機能
  - タスクの生成/削除
  - タスクの起動/終了
  - 優先度の変更

タスク管理機能:サービスコール(1)

- タスクの生成(静的API)
  - CRE_TSK(ID tskid, {ATR tskatr, VP_INT exinf, FP task, PRI itskpri, SIZE stksz, VP stk});
- タスクの生成
  - ER ercd = cre_tsk(ID tskid, T_CTSK* pk_ctsk);
  - ER_ID tskid = acre_tsk(T_CTSK* pk_ctsk);
- タスクの削除
  - ER ercd = del_tsk(ID tskid);

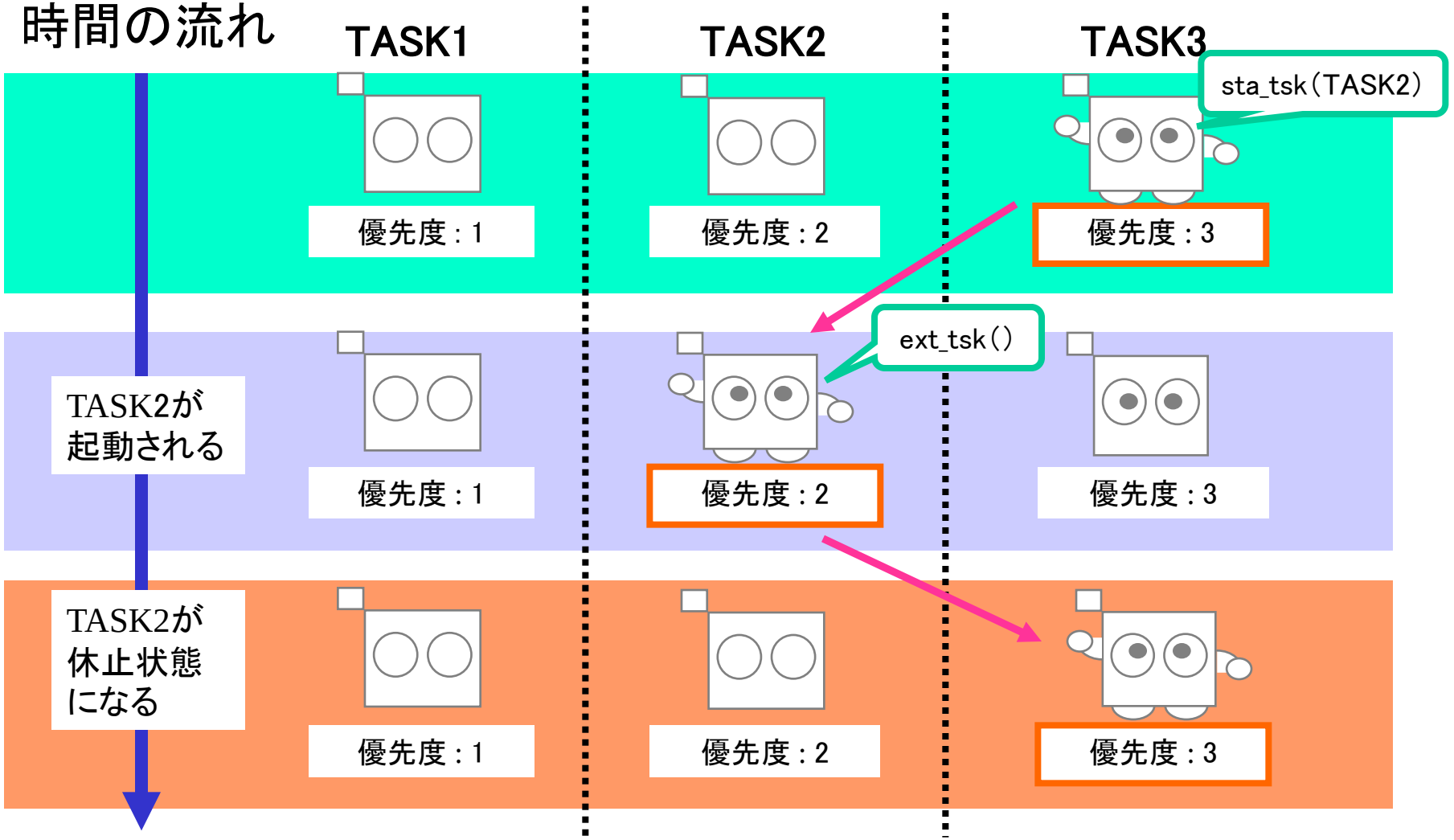
タスク管理機能:サービスコール(2)

- タスクの起動
  - ER ercd = act_tsk(ID tskid);
  - ER ercd = iact_tsk(ID tskid);
- タスク起動要求のキャンセル
  - ER_UINT actcnt = can_act(ID tskid);
- タスクの起動(起動コード指定)
  - ER ercd = sta_tsk(ID tskid, VP_INT stacd);
- 自タスクの終了
  - void ext_tsk();
  - void exd_tsk();

タスク管理機能:サービスコール(3)

- タスクの強制終了
  - ER ercd = ter_tsk(ID tskid);
- タスク優先度の変更
  - ER ercd = chg_pri(ID tskid, PRI tskpri);
- タスク優先度の参照
  - ER ercd = get_pri(ID tskid, PRI* p_tskpri);
- タスクの状態参照
  - ER ercd = ref_tsk(ID tskid, T_RTSK* pk_rtsk);
  - ER ercd = ref_tst(ID tskid, T_RTST* pk_rtst);

sta_tskとext_tskの動作例



タスク付属同期機能

- 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
  - タスクの起床待ちと起床
  - タスクの強制待ちと解除

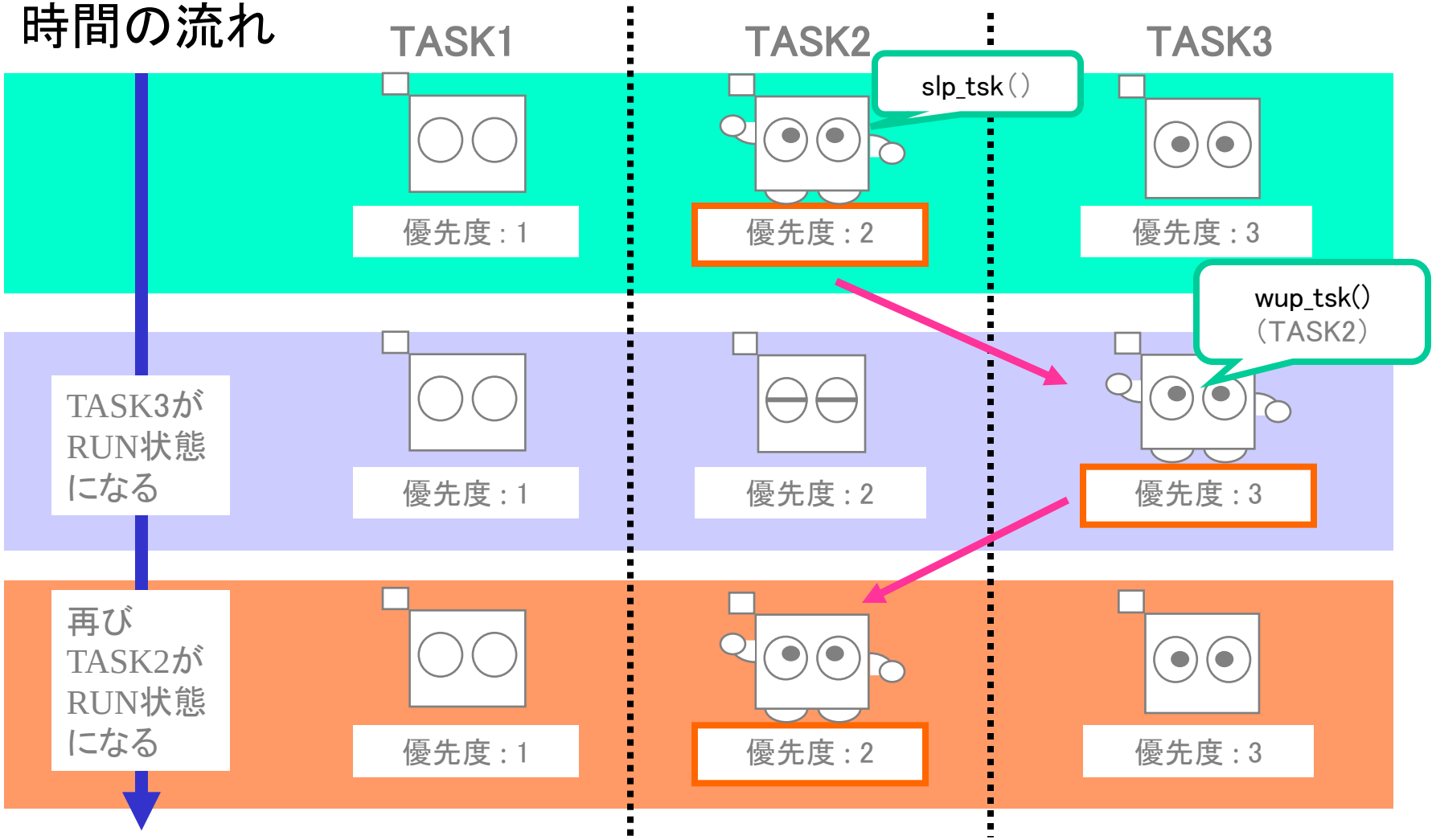
タスク付属同期機能: サービスコール(1)

- 起床待ち
  - ER ercd = slp_tsk();
  - ER ercd = tslp_tsk(TMO tmout);
- タスクの起床
  - ER ercd = wup_tsk(ID tskid);
  - ER ercd = iwup_tsk(ID tskid);
- タスク起床要求のキャンセル
  - ER_UINT wupcnt = can_wup(ID tskid);

タスク付属同期機能: サービスコール(2)

- 待ち状態の強制解除
  - ER ercd = rel_wai(ID tskid);
  - ER ercd = irel_wai(ID tskid);
- 強制待ち状態への移行
  - ER ercd = sus_tsk(ID tskid);
- 強制待ち状態からの再開
  - ER ercd = rsm_tsk(ID tskid);
  - ER ercd = frsm_tsk(ID tskid);
- 自タスクの遅延
  - ER ercd = dly_tsk(RELTIM dlytim);

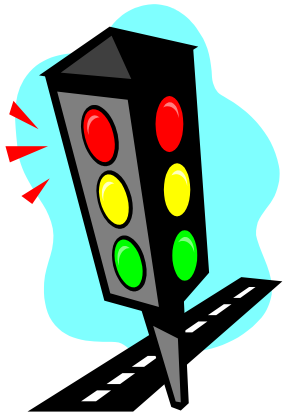
slp_tskとwup_tskの動作例



同期・通信機能

- 複数のタスクの動作の同期を取ったり、タスク間通信を行う時に使われる機能
  - タスク間の処理の同期
  - タスクで使用する資源の排他制御
  - ある事象をタスクに通知する
  - 他のタスクにデータを送信する
  - 代表的なオブジェクトに、セマフォ、イベントフラグ、メールボックスなどがある
- 使用例
  - 複数のタスクで1つの資源を共用して使いたい場合
  - 複数の割込み事象を1つのタスクで割込み要因別に処理したい場合

セマフォ



- 機能
  - タスク間での資源（I/Oやメモリ）の排他的使用に利用
  - 資源の使用側タスクと返却側タスクとの間で通信を行う
  - セマフォカウンタの設定により、資源の共有数を指定できる
  - 待ちタスクの待ち方はFIFO順または優先度順で指定

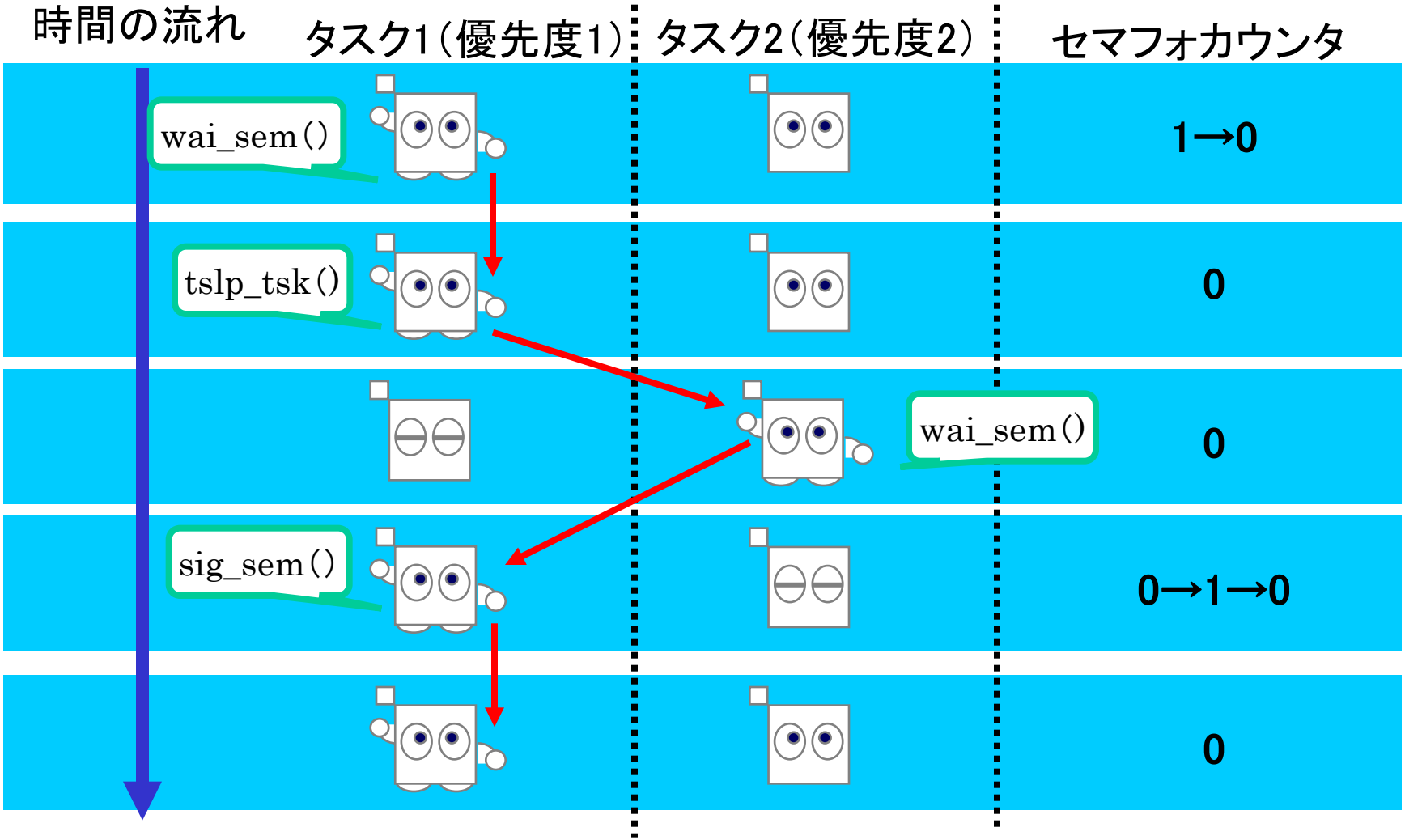
セマフォ:サービスコール(1)

- セマフォの生成(静的API)
  - CRE_SEM(ID semid, {ATR sematr, UINT isemcnt, UINT maxsem});
- セマフォの生成
  - ER ercd = cre_sem(ID semid, T_CSEM* pk_csem);
  - ER_ID semid = acre_sem(T_CSEM* pk_csem);
- セマフォの削除
  - ER ercd = del_sem(ID semid);

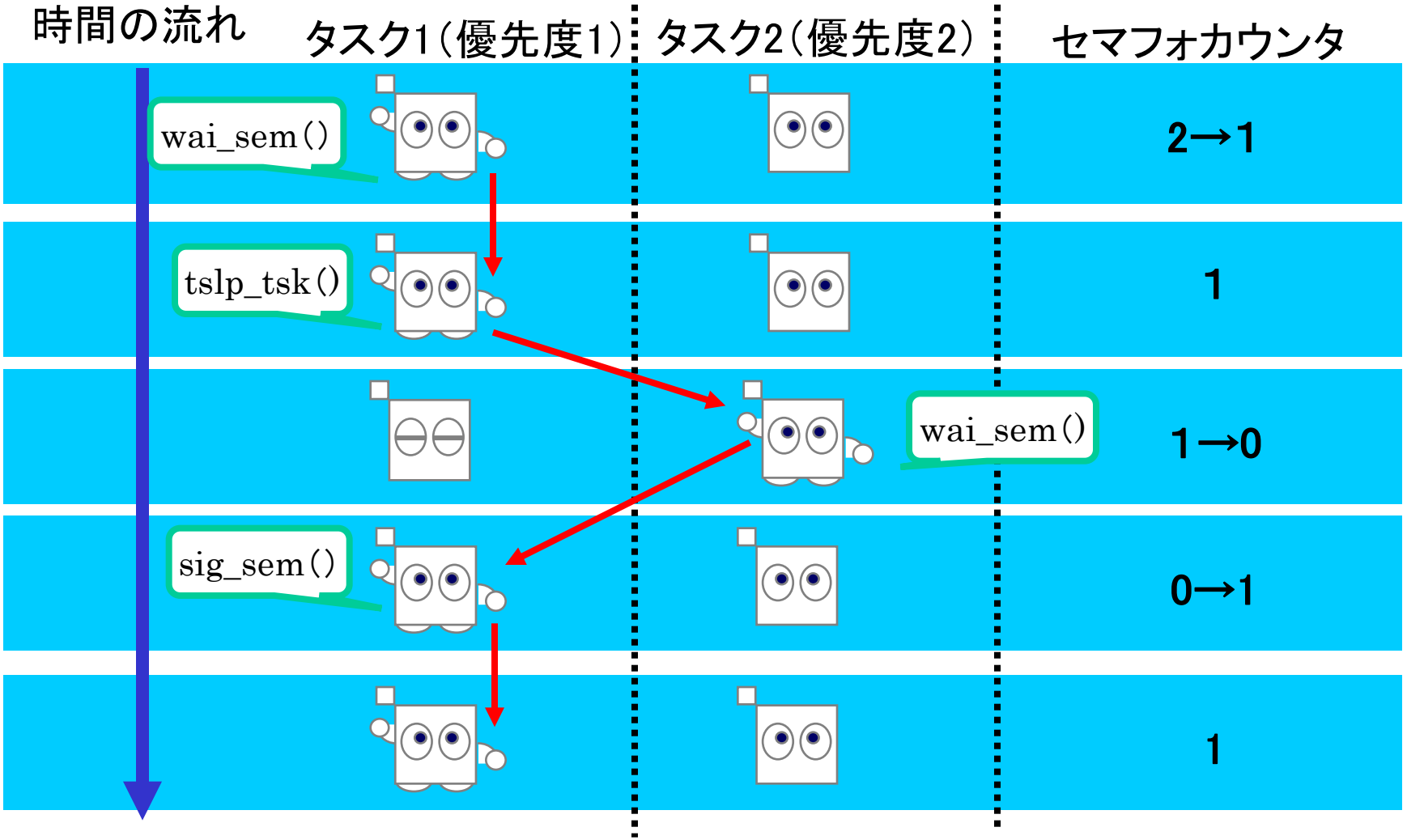
セマフォ:サービスコール(2)

- セマフォ資源の返却
  - ER ercd = sig_sem(ID semid);
  - ER ercd = isig_sem(ID semid);
- セマフォ資源の獲得
  - ER ercd = wai_sem(ID semid);
  - ER ercd = pol_sem(ID semid);
  - ER ercd = twai_sem(ID semid, TMO tmout);
- セマフォの状態参照
  - ER ercd = ref_sem(ID semid, T_RSEM* pk_rsem);

セマフォ　動作例－1


時間の流れ
タスク1（優先度1）
タスク2（優先度2）
セマフォカウンタ
wai_sem()
1→0
tslp_tsk()
0
wai_sem()
0
sig_sem()
0→1→0
0

セマフォ　動作例－2


時間の流れ
タスク1（優先度1）
タスク2（優先度2）
セマフォカウンタ
wai_sem()
2→1
tslp_tsk()
1
wai_sem()
1→0
sig_sem()
0→1
1

イベントフラグ

- 機能
  - 事象の通知をタスク/ハンドラ間で通信する
  - ビットパターンを事象に割り当てることで利用
  - OR待ちやAND待ちなどの事象待ちが可能
  - パターンが一致すれば同時に複数のタスクを待ち状態から復帰可能

イベントフラグ:サービスコール(1)

- イベントフラグの生成(静的API)
  - CRE_FLG(ID flgid, {ATR flgatr, FLGPTN iflgptn});
- イベントフラグの生成
  - ER ercd = cre_flg(ID flgid, T_CFLG* pk_cflg);
  - ER_ID flgid = acre_flg(T_CFLG* pk_cflg);
- イベントフラグの削除
  - ER ercd = del_flg(ID flgid);

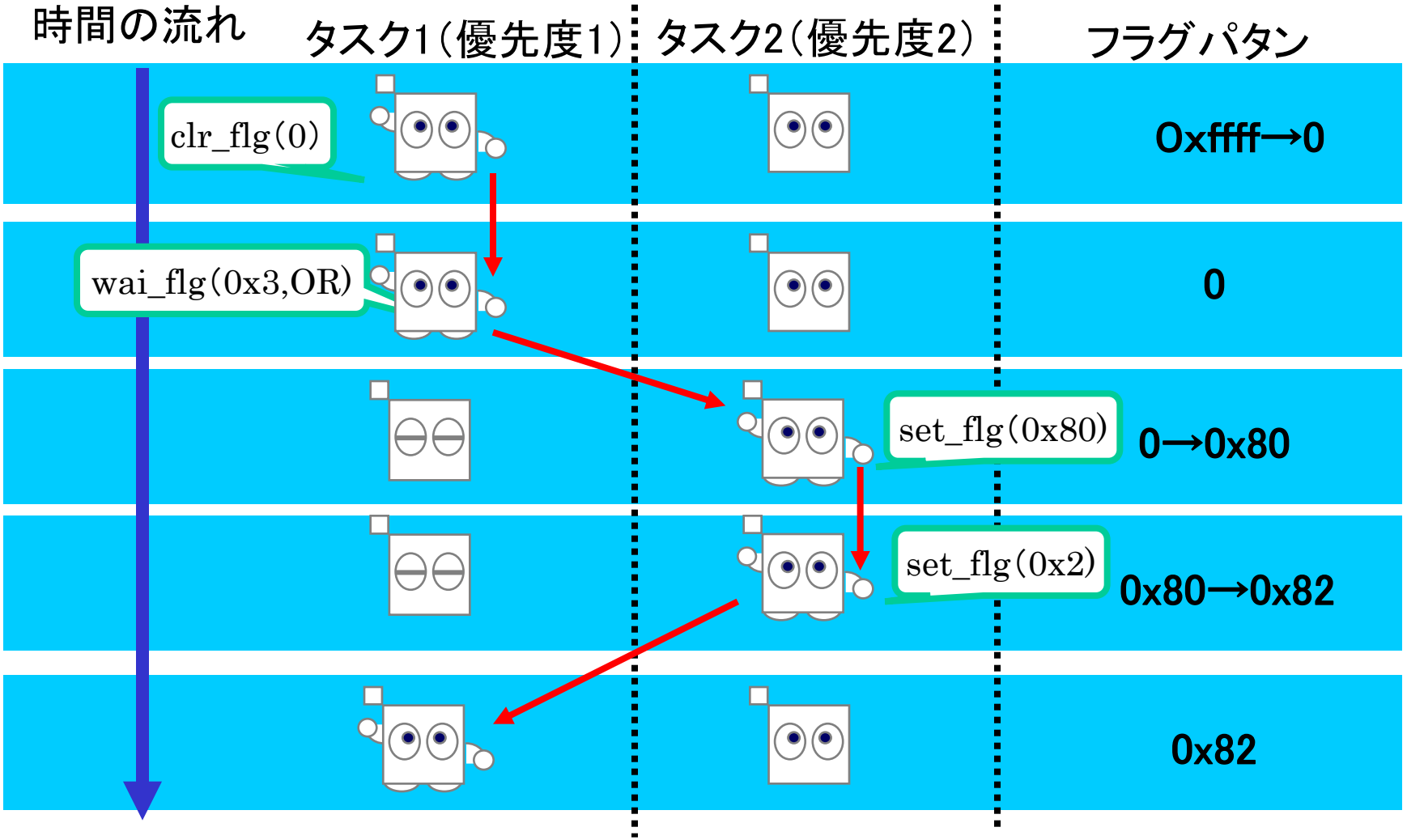
イベントフラグ:サービスコール(2)

- イベントフラグのセット
  - ER ercd = set_flg(ID flgid, FLGPTN setptn);
  - ER ercd = iset_flg(ID flgid, FLGPTN setptn);
- イベントフラグのクリア
  - ER ercd = clr_flg(ID flgid, FLGPTN clrptn);

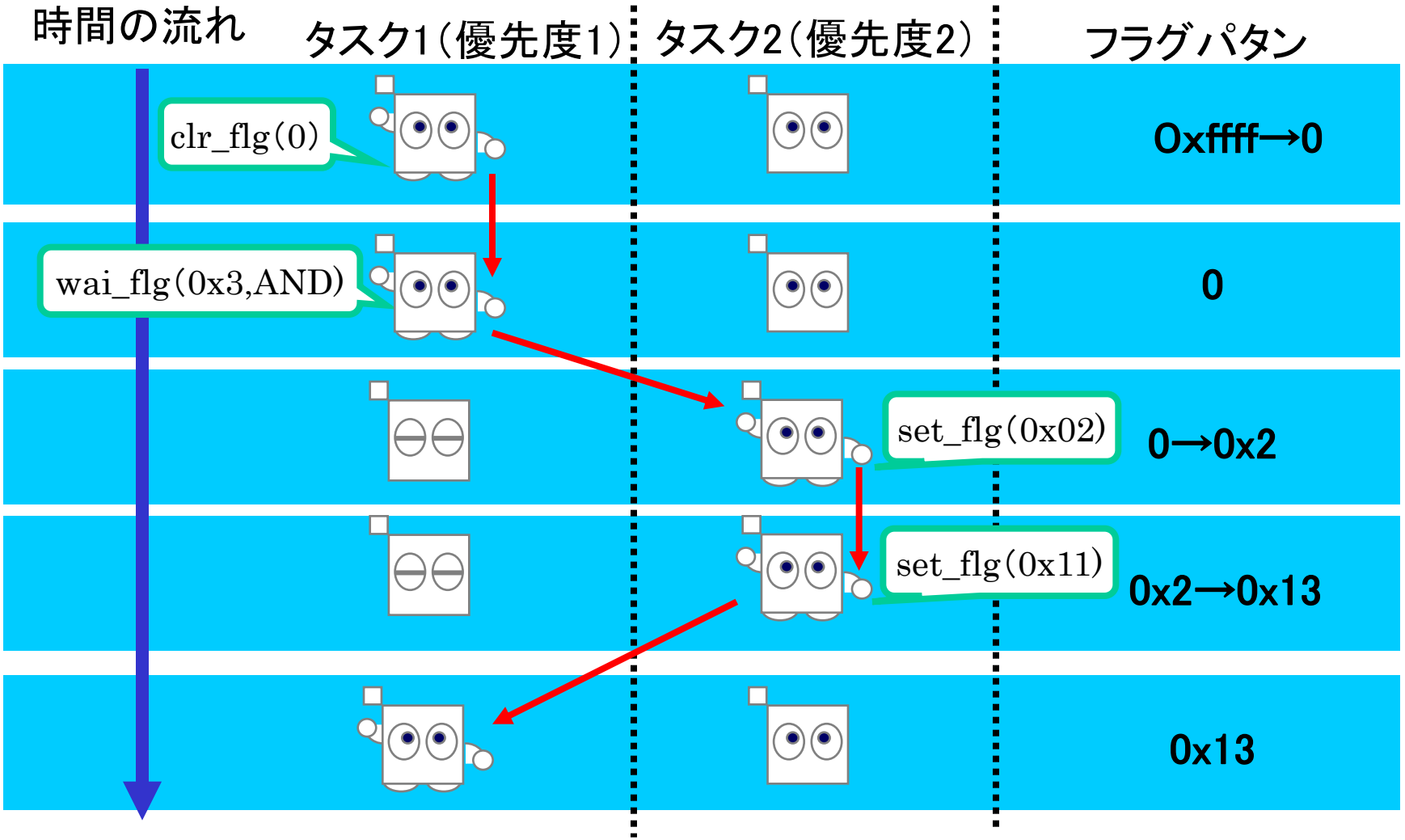
イベントフラグ:サービスコール(3)

- イベントフラグ待ち
  - ER ercd = wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN* p_flgptn);
  - ER ercd = pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN* p_flgptn);
  - ER ercd = twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN* p_flgptn, TMO tmout);
- イベントフラグの状態参照
  - ER ercd = ref_flg(ID flgid, T_RFLG* pk_rflg);

イベントフラグ　動作例－1



イベントフラグ　動作例－2



データキュー

- 機能
  - タスク間で1ワードのメッセージを通信
  - 送信データはデータキュー領域にコピー
  - メッセージの登録順序はFIFO順
  - 待ちタスクの待ち方はFIFO順または優先度順で指定

データキュー:サービスコール(1)

- データキューの生成(静的API)
  - CRE_DTQ(ID dtqid, {ATR dtqatr, UINT dtqcnt, VP dtq});
- データキューの生成
  - ER ercd = cre_dtq(ID dtqid, T_CDTQ* pk_cdtq);
  - ER_ID dtqid = acre_dtq(T_CDTQ* pk_cdtq);
- データキューの削除
  - ER ercd = del_dtq(ID dtqid);

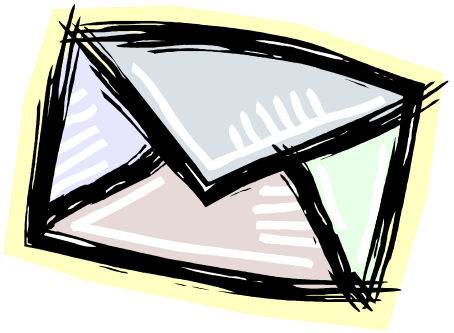
データキュー:サービスコール(2)

- データキューへの送信
  - ER ercd = snd_dtq(ID dtqid, VP_INT data);
  - ER ercd = psnd_dtq(ID dtqid, VP_INT data);
  - ER ercd = ipsnd_dtq(ID dtqid, VP_INT data);
  - ER ercd = tsnd_dtq(ID dtqid, VP_INT data, TMO tmout);
- データキューへの強制送信
  - ER ercd = fsnd_dtq(ID dtqid, VP_INT data);
  - ER ercd = ifsnd_dtq(ID dtqid, VP_INT data);

データキュー:サービスコール(3)

- データキューからの受信
  - ER ercd = rcv_dtq(ID dtqid, VP_INT* p_data);
  - ER ercd = prcv_dtq(ID dtqid, VP_INT* p_data);
  - ER ercd = trcv_dtq(ID dtqid, VP_INT* p_data, TMO tmout);
- データキューの状態参照
  - ER ercd = ref_dtq(ID dtqid, T_RDTQ* pk_rdtq);

メールボックス



- 機能
  - タスク間でメッセージ領域のアドレスを通信
  - 送信側から受信側へメッセージはコピーされない
  - メッセージの登録順序はFIFO順または優先度順で指定
  - 待ちタスクの待ち方はFIFO順または優先度順で指定

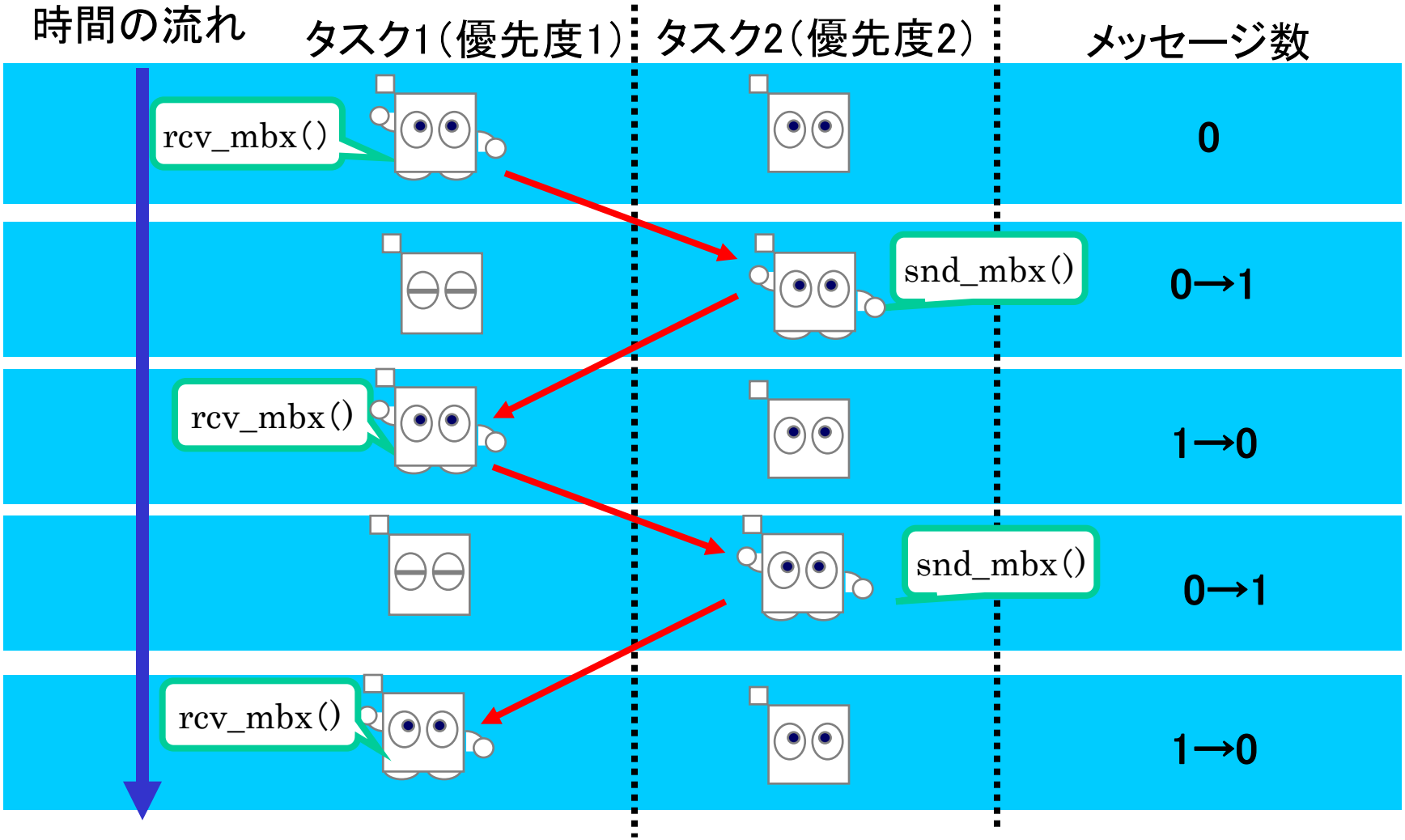
メールボックス:サービスコール(1)

- メールボックスの生成(静的API)
  - CRE_MBX(ID mbxid, {ATR mbxatr, PRI maxmpri, VP mprihd});
- メールボックスの生成
  - ER ercd = cre_mbx(ID mbxid, T_CMBX* pk_cmbx);
  - ER_ID mbxid = acre_mbx(T_CMBX* pk_cmbx);
- メールボックスの削除
  - ER ercd = del_mbx(ID mbxid);

メールボックス:サービスコール(2)

- メールボックスへの送信
  - ER ercd = snd_mbx(ID mbxid, T_MSG* pk_msg);
- メールボックスからの受信
  - ER ercd = rcv_mbx(ID mbxid, T_MSG** ppk_msg);
  - ER ercd = prcv_mbx(ID mbxid, T_MSG** ppk_msg);
  - ER ercd = trcv_mbx(ID mbxid, T_MSG** ppk_msg, TMO tmout);
- メールボックスの状態参照
  - ER ercd = ref_mbx(ID mbxid, T_RMBX* pk_rmbx);

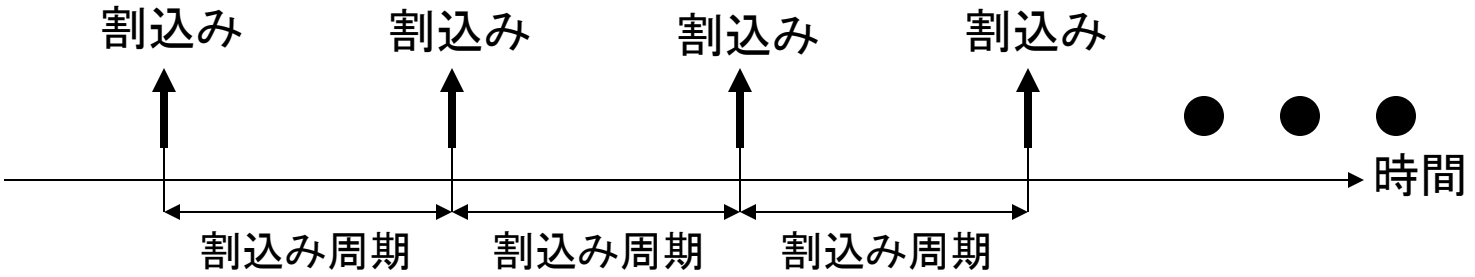
メールボックス　動作例－1

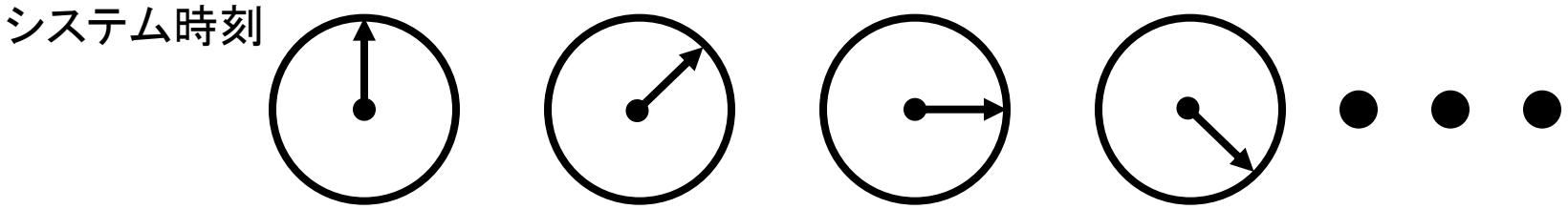


時間管理機能

- システム時刻
  - ハードウェアからの割込みを基準にして時間に同期した処理を行う機能
- タイムアウト処理機能
- タイムイベント機能
  - 周期ハンドラ:一定間隔でハンドラを呼出す機能
  - アラーム・ハンドラ:指定した時刻に一回だけハンドラを呼出す機能
  - オーバーラン・ハンドラ:タスクが指定時間以上実行状態となった場合にハンドラを呼出す機能

システム時刻



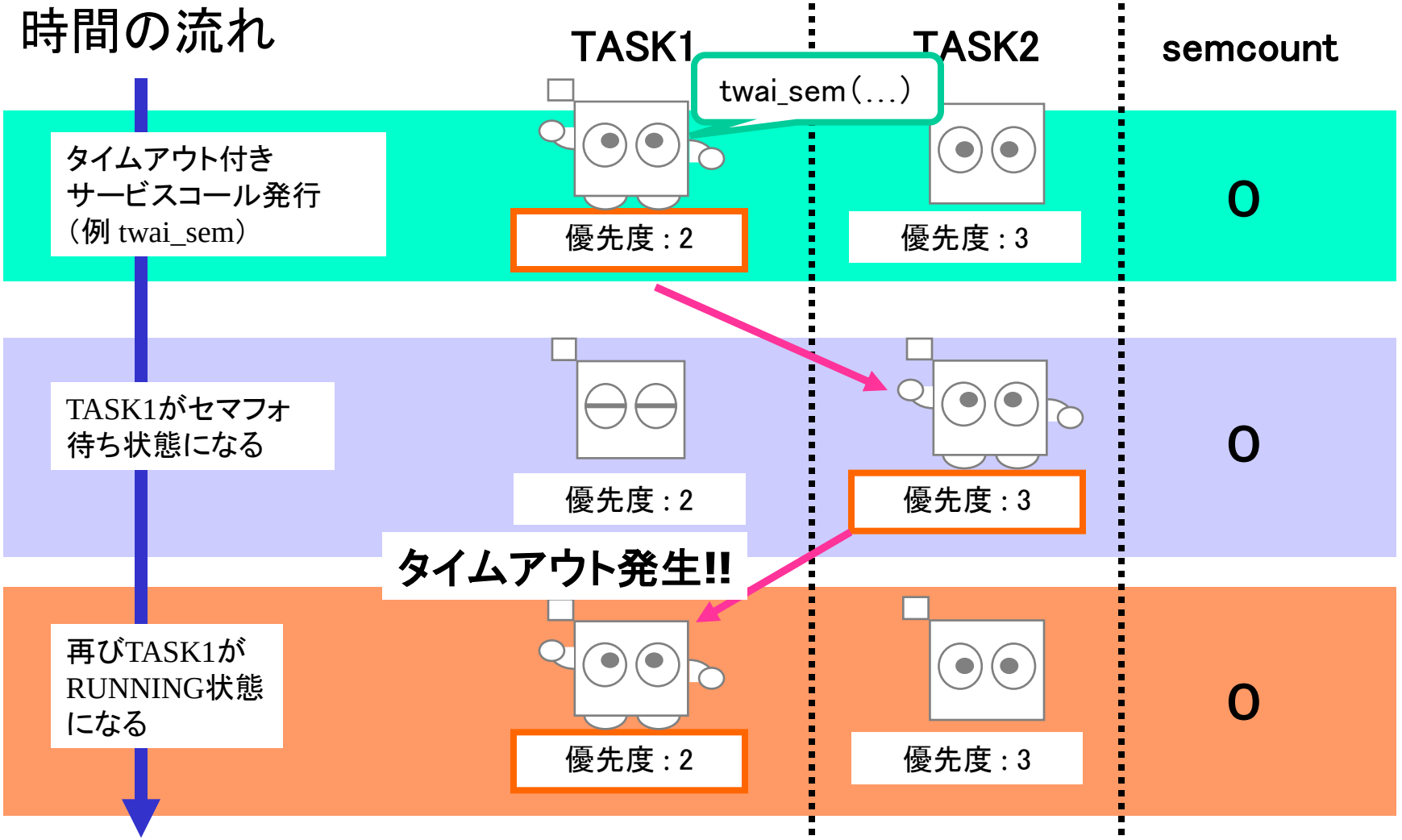


割込みのたびにシステムの時計が進む

システム時刻:サービスコール

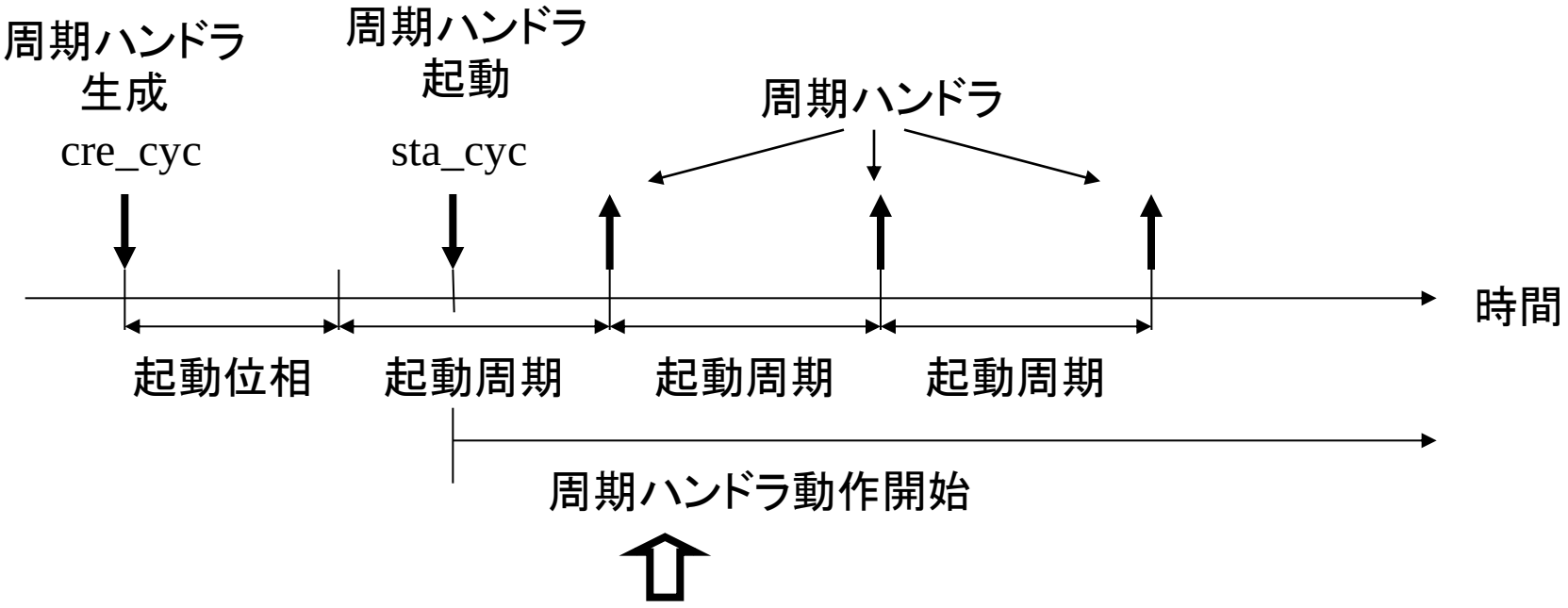
- システム時刻の設定
  - ER ercd = set_tim(SYSTIM* p_systim);
- システム時刻の参照
  - ER ercd = get_tim(SYSTIM* p_systim);

タイムアウト処理機能


時間の流れ
TASK1
TASK2
semcount
twai_sem(...)
タイムアウト付き
サービスコール発行
（例 twai_sem）
優先度：2
優先度：3
0
TASK1がセマフォ
待ち状態になる
優先度：2
優先度：3
0
タイムアウト発生!!
再びTASK1が
RUNNING状態
になる
優先度：2
優先度：3
0

周期ハンドラ(1)

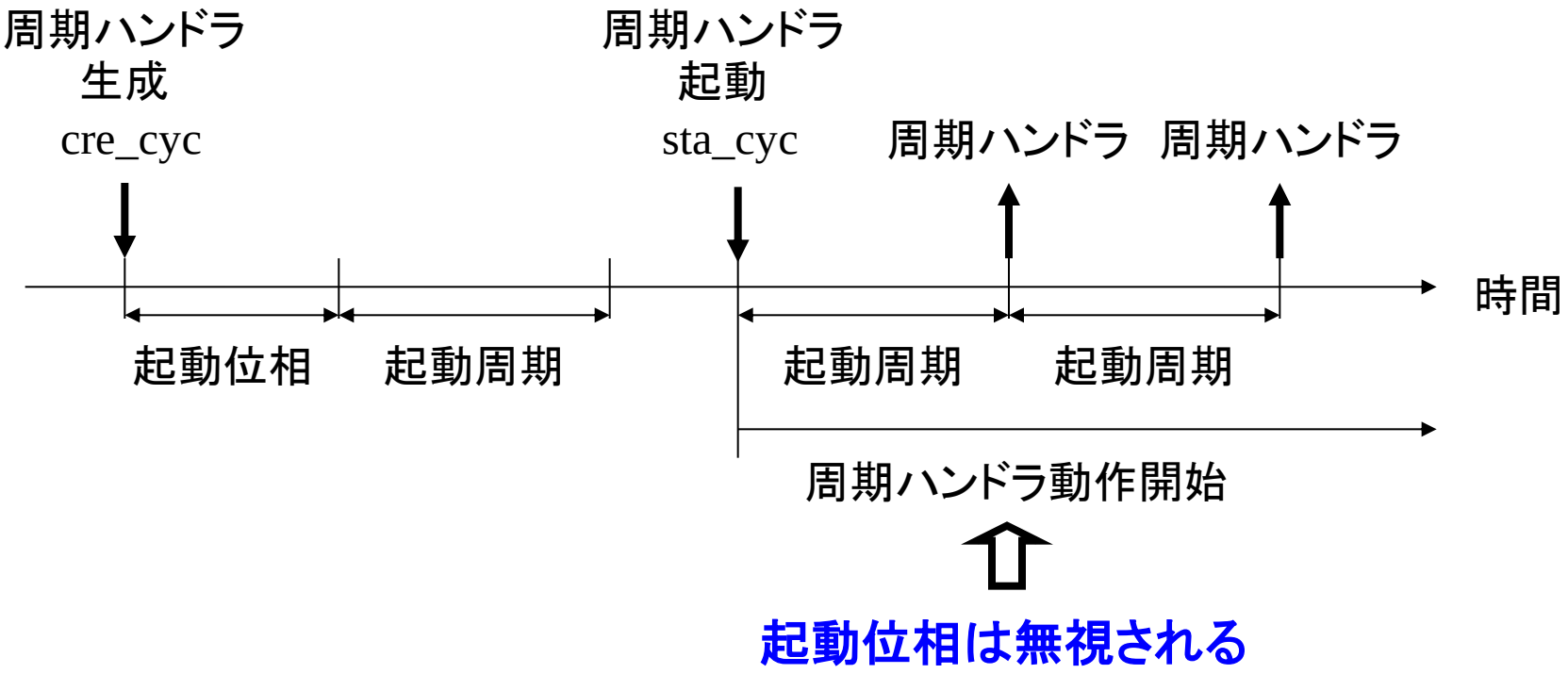
起動位相保存有りの場合



起動位相が考慮される

周期ハンドラ(2)

起動位相保存無しの場合



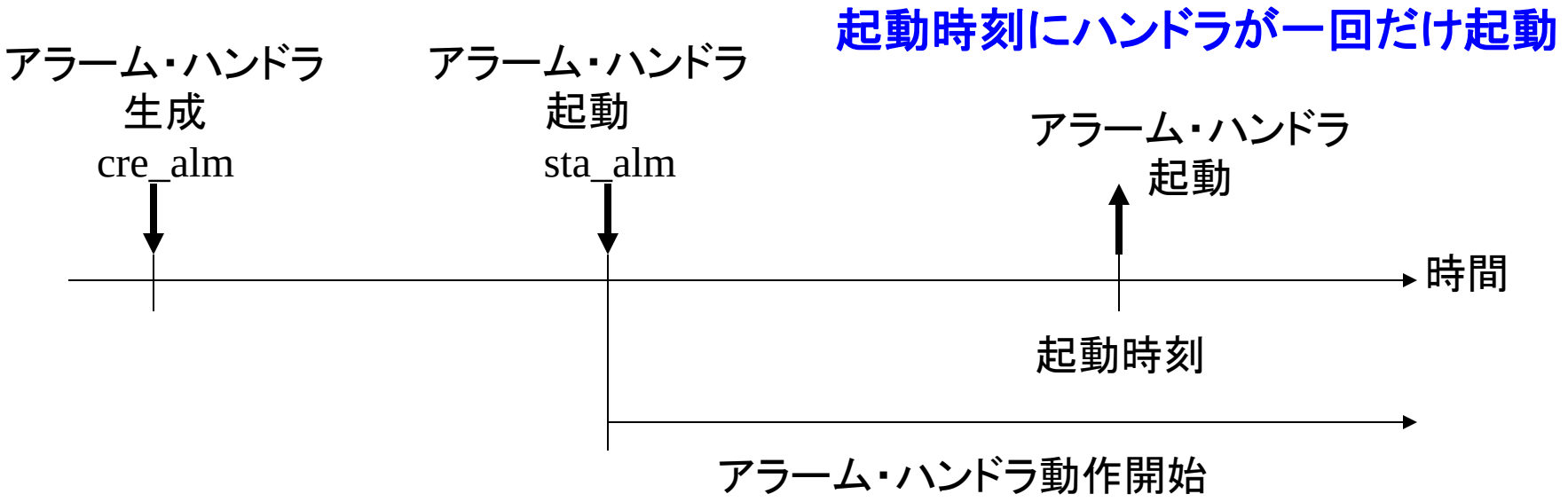
周期ハンドラ:サービスコール(1)

- 周期ハンドラの生成(静的API)
  - CRE_CYC(ID cycid, {ATR cycatr, VP_INT exinf, FP cychdr, RELTIM cyctim, RELTIM cycphs});
- 周期ハンドラの生成
  - ER ercd = cre_cyc(ID cycid, T_CCYC* pk_ccyc);
  - ER_ID cycid = acre_cyc(T_CCYC* pk_ccyc);
- 周期ハンドラの削除
  - ER ercd = del_cyc(ID cycid);

周期ハンドラ:サービスコール(2)

- 周期ハンドラの動作開始
  - ER ercd = sta_cyc(ID cycid);
- 周期ハンドラの動作停止
  - ER ercd = stp_cyc(ID cycid);
- 周期ハンドラの状態参照
  - ER ercd = ref_cyc(ID cycid, T_RCYC* pk_rcyc);

アラームハンドラ


起動時刻にハンドラが一回だけ起動
アラーム・ハンドラ
生成
cre_alm
アラーム・ハンドラ
起動
sta_alm
アラーム・ハンドラ
起動
時間
起動時刻
アラーム・ハンドラ動作開始

アラームハンドラ: サービスコール(1)

- アラームハンドラの生成(静的API)
  - CRE_ALM(ID almid, {ATR almatr, VP_INT exinf, FP almhdr});
- アラームハンドラの生成
  - ER ercd = cre_alm(ID almid, T_CALM* pk_calm);
  - ER_ID almid = acre_alm(T_CALM* pk_calm);
- アラームハンドラの削除
  - ER ercd = del_alm(ID almid);

アラームハンドラ: サービスコール(2)

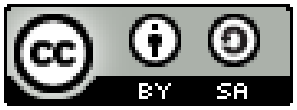
- アラームハンドラの動作開始
  - ER ercd = sta_alm(ID almid, RELTIM almtim);
- アラームハンドラの動作停止
  - ER ercd = stp_alm(ID almid);
- アラームハンドラの状態参照
  - ER ercd = ref_alm(ID almid, T_RALM* pk_ralm);

【実習】ITRON初級編テキスト「プログラミング演習(1)」

著者　　TRON Forum

本テキストは、クリエイティブ・コモンズ 表示 - 継承 4.0 国際 ライセンスの下に提供されています。

https://creativecommons.org/licenses/by-sa/4.0/deed.ja



Copyright ©2016 TRON Forum

【ご注意およびお願い】

1.本テキストの中で第三者が著作権等の権利を有している箇所については、利用者の方が当該第三者から利用許諾を得てください。

2.本テキストの内容については、その正確性、網羅性、特定目的への適合性等、一切の保証をしないほか、本テキストを利用したことにより損害が生じても著者は責任を負いません。

3.本テキストをご利用いただく際、可能であれば office@tron.org までご利用者のお名前、ご所属、ご連絡先メールアドレスをご連絡いただければ幸いです。