μITRON4.0仕様の徹底解説
パート2
「μITRON4.0仕様の概念と割込み管理」

三菱電機セミコンダクタシステム株式会社
村木宏行

基本的な用語の定義

- タスクと自タスク
  - プログラムの並行実行の単位を「タスク」
  - **サービスコールを呼び出したタスクを「自タスク」**
- ディスパッチとディスパッチャ
  - タスクを切り替えることを「ディスパッチ」
  - ディスパッチ処理をする機構を「ディスパッチャ」
- 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」
  - スケジューリングを行う機構を「スケジューラ」
- **コンテキスト**
  - **プログラムを実行する環境**
- **優先順位**
  - **処理が実行される順序を決める優先関係**

処理単位とコンテキスト

- **割込みハンドラ**
  - **割込みサービスルーチン**
- **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**

（非タスクコンテキスト）

- **CPU例外ハンドラ**
- **拡張サービスコールルーチン**
- **タスク**
  - **タスク例外処理ルーチン**

（タスクコンテキスト）

CPU例外ハンドラと拡張サービスコールルーチンのコンテキスト

CPU例外ハンドラ

- CPU例外がタスクなどのタスクコンテキストで発生した場合は実装定義
- CPU例外が割込みハンドラなどの非タスクコンテキストで発生した場合は非タスクコンテキスト

拡張サービスコールルーチン

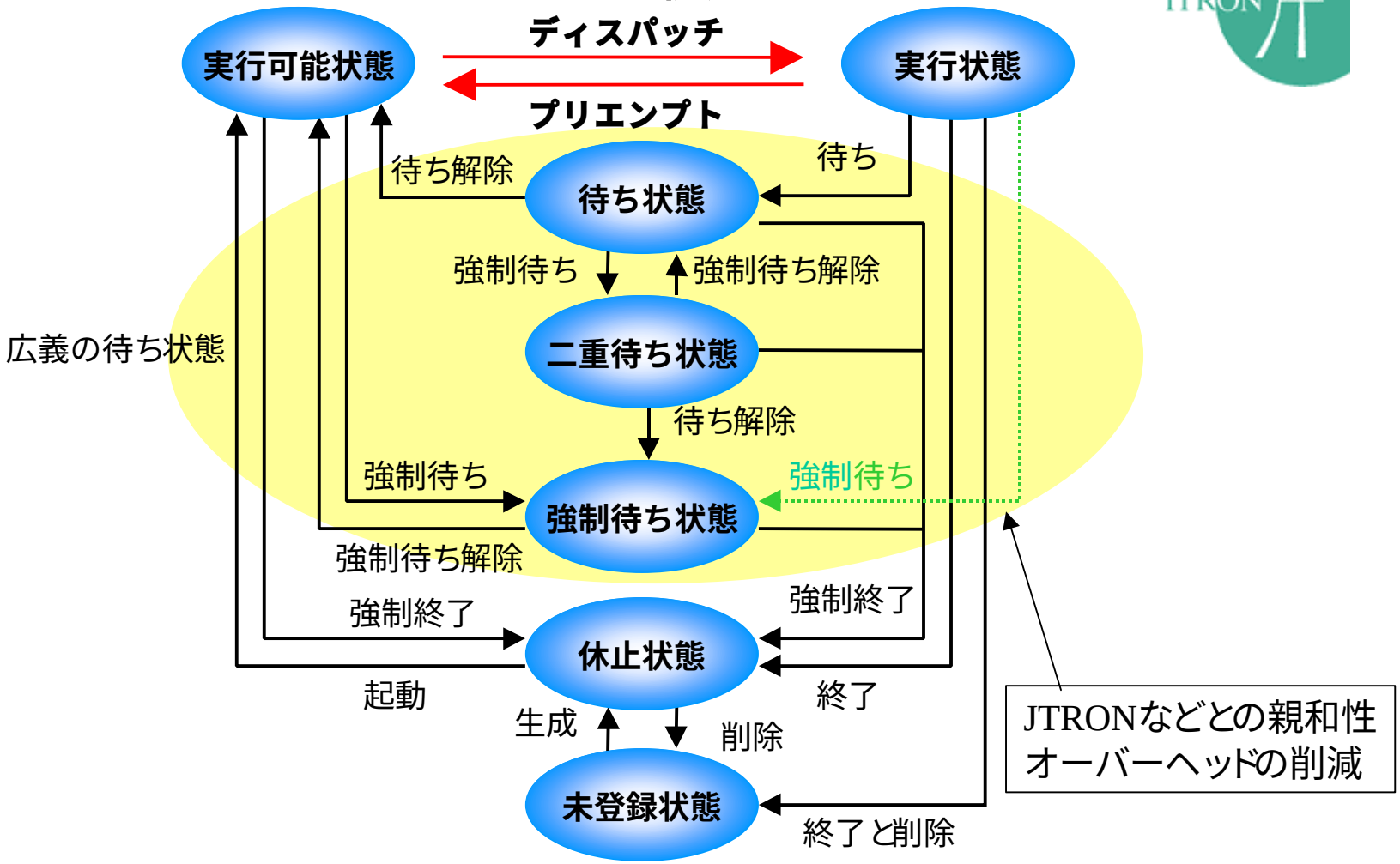
- タスクなどのタスクコンテキストから呼び出された場合はタスクコンテキスト
- 割込みハンドラなどの非タスクコンテキストから呼び出された場合は非タスクコンテキスト

タスク状態

それぞれの意味や内容はμITRON3.0仕様とほぼ同じ
英語名を形容詞形に変更

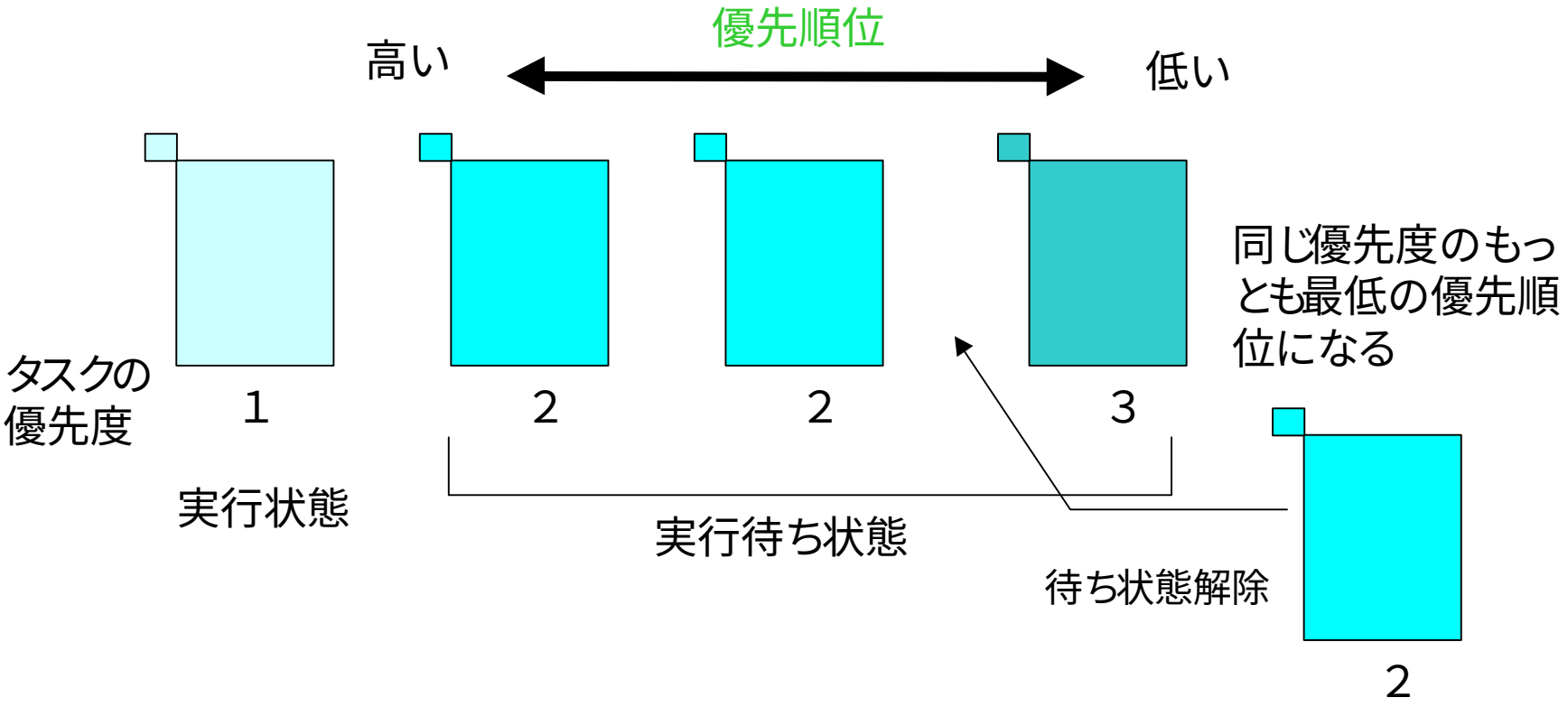
- 実行状態（RUNNING）
- 実行可能状態（READY）
- 広義の待ち状態
  - 待ち状態（WAITING）
  - 強制待ち状態（SUSPENDED）
  - 二重待ち状態（WAITING-SUSPENDED）
- 休止状態（DORMANT）
- 未登録状態（NON-EXISTENT）

タスク状態


実行可能状態
ディスパッチ
プリエンプト
実行状態
待ち解除
待ち状態
待ち
強制待ち
強制待ち解除
広義の待ち状態
二重待ち状態
待ち解除
強制待ち
強制待ち
強制待ち状態
強制待ち解除
強制終了
強制終了
休止状態
起動
終了
生成
削除
未登録状態
終了と削除
JTRONなどとの親和性
オーバーヘッドの削減

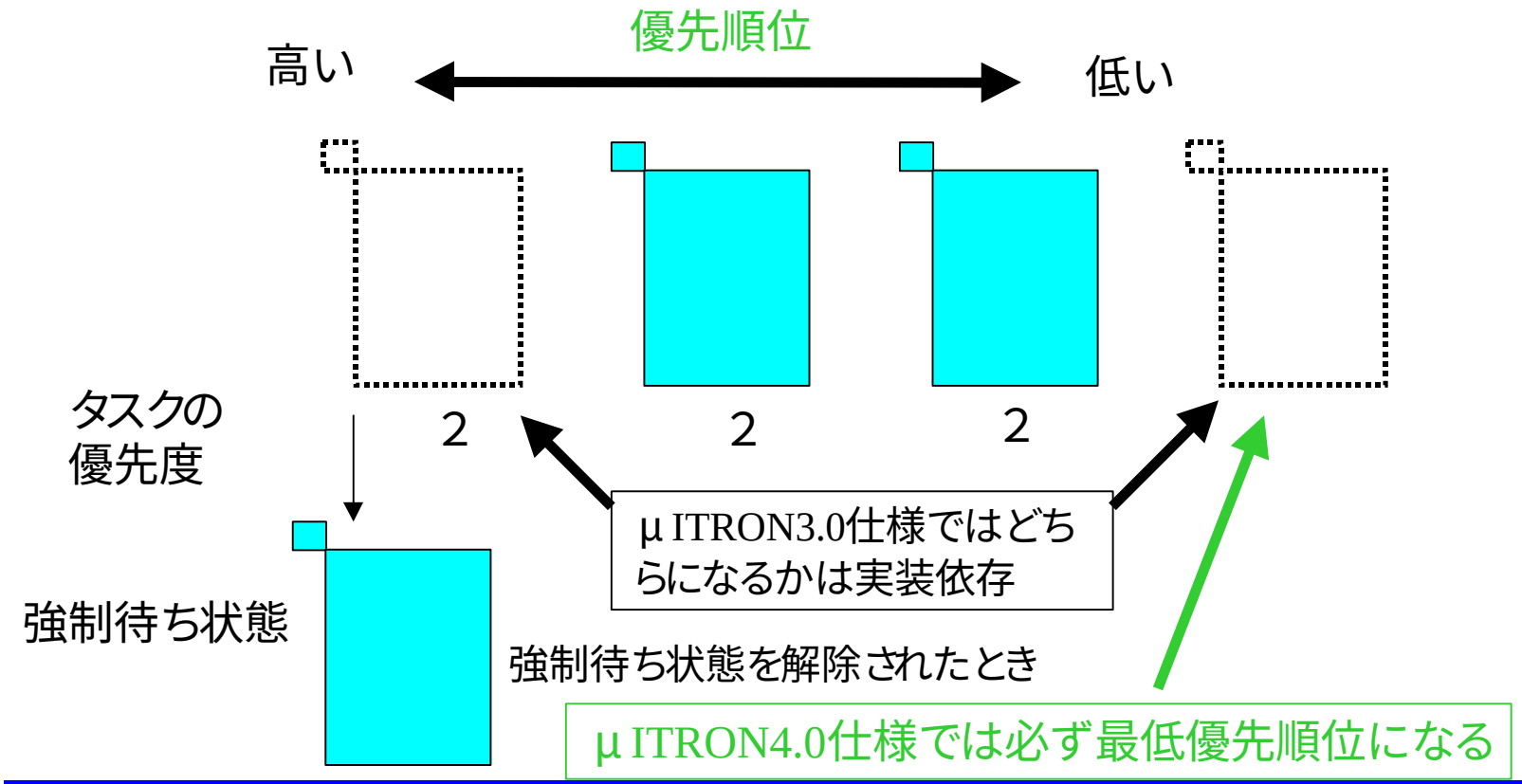
スケジューリング規則 (1)

- 優先度ベーススケジューリング方式
- FCFS (ファーストカムファーストサービス) 方式

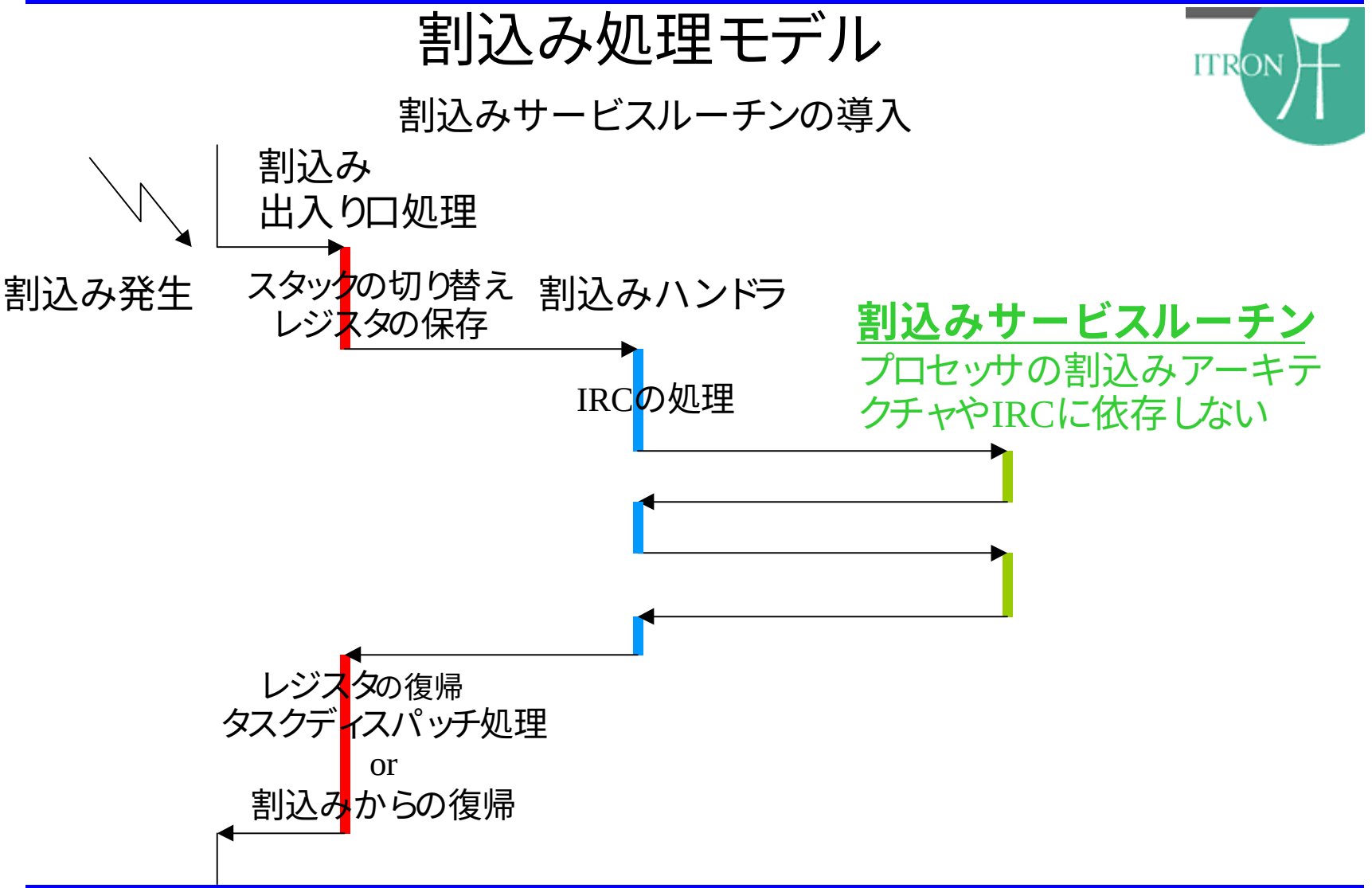


スケジューリング規則 (2)

μITRON3.0仕様との相違点



割込み処理モデル


割込みサービスルーチンの導入
割込み
出入り口処理
割込み発生
スタックの切り替え
レジスタの保存
割込みハンドラ
IRCの処理
割込みサービスルーチン
プロセッサの割込みアーキテクチャやIRCに依存しない
レジスタの復帰
タスクディスパッチ処理
or
割込みからの復帰

割込み管理機能

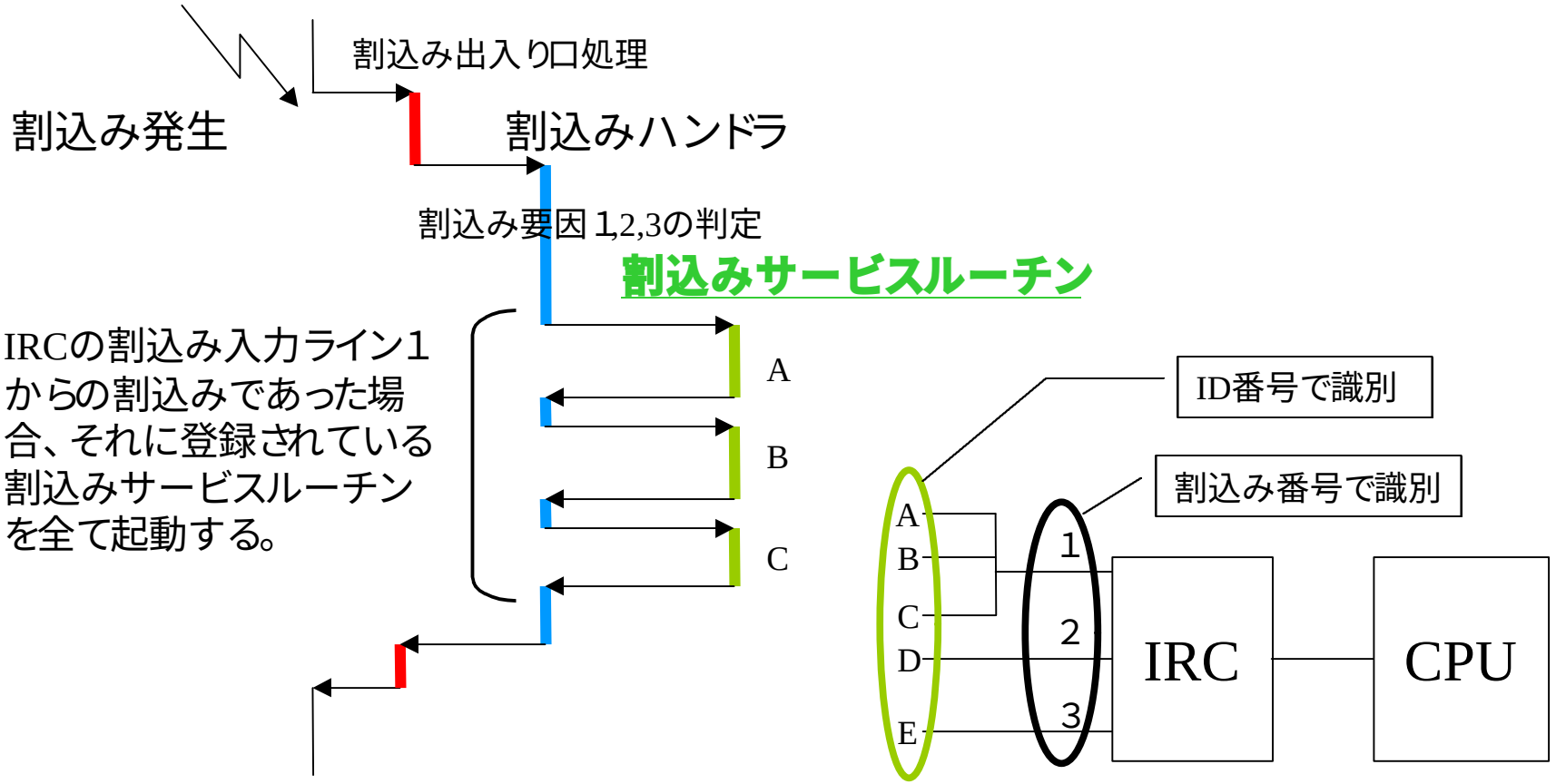
サービスコール名	機能
DEF_INH	割込みハンドラの定義(静的API)【S】
def_inh	割込みハンドラの定義
ATT_ISR	割込みサービスルーチンの追加(静的API)【S】
cre_isr	割込みサービスルーチンの生成
acre_isr	割込みサービスルーチンの生成（ID番号自動割付）
del_isr	割込みサービスルーチンの削除
ref_isr	割込みサービスルーチンの状態参照
dis_int	割込みの禁止
ena_int	割込みの許可
chg_ixx	割込みマスクの変更
get_ixx	割込みマスクの参照

【S】:スタンダードプロファイル

割込みハンドラと割込みサービスルーチン

＝ 実装の例 (1) ＝

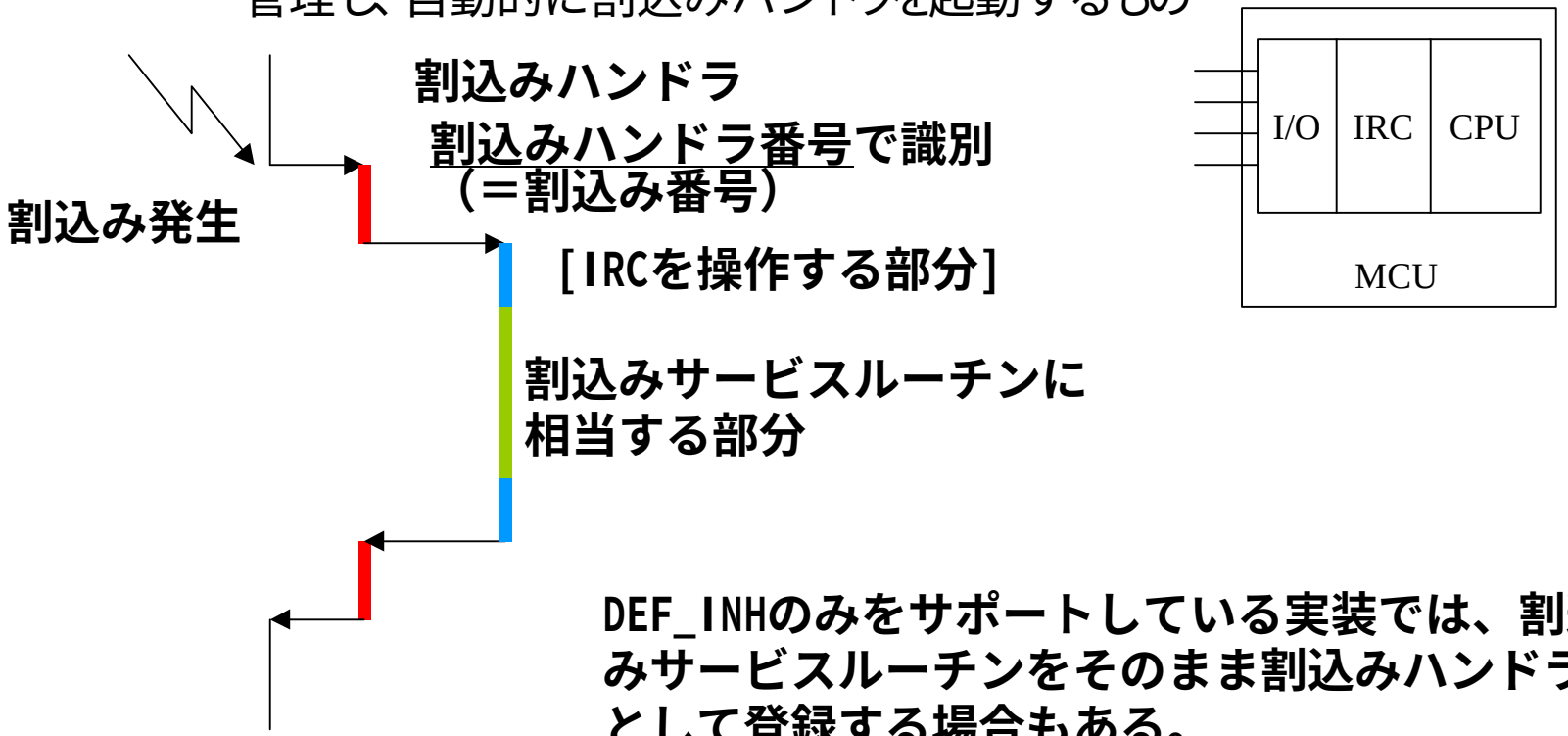
複数の割込み要求が1つの割込み信号で入ってくる例



割込みハンドラと割込みサービスルーチン

＝　実装の例（2）＝

シングルチップマイコンなど、各割込み要因をベクタテーブルで管理し、自動的に割込みハンドラを起動するもの



DEF_INHのみをサポートしている実装では、割込みサービスルーチンをそのまま割込みハンドラとして登録する場合もある。

カーネル管理外割込みハンドラ

- カーネル管理外の割込みハンドラとは
  ある優先度より高い優先度を持つ割込み（禁止できない割込みを含む）でサービスコールを呼び出すことができない
- μITRON3.0仕様では特に規定していない
  実装独自で定義している
- μITRON4.0仕様ではカーネル管理外の割込みはサービスコールで禁止できないと明記した
  ＝loc_cpuサービスコールの対象外

例外処理

CPU例外処理ハンドラ

CPU例外によって起動されるハンドラ

タスク例外処理ルーチンを起動するなどして、例外処理をおこなう

以下の処理が可能であればよい

（その他は実装に依存する部分が多いため、規定しない）

- CPU例外が発生したコンテキストや状態が取り出せること
  - CPU例外が発生した処理でsns_*yyy*を発行したのと同等の結果を知ることができる
- CPU例外が発生したタスクIDの読み出し
- タスク例外処理の要求

DEF_EXCまたはdef_excで定義

- μITRON3.0仕様ではこのシステムコールでCPU例外以外の例外に対する処理も定義できることになっていた

タスク例外処理ルーチン

詳細はパート3で説明

何か例外的な事象が起きたことに対応するためのルーチン

処理の優先順位

処理の優先順位を以下のように定義

(1) 割込みハンドラ、タイムイベントハンドラ、CPU例外ハンドラ
(2) ディスパッチャ（カーネル処理の一部）
(3) タスク

割込みハンドラなどの途中では切り替えが起こらず、すべての割込みハンドラが終了したときにディスパッチが起こる

μITRON3.0仕様以前は遅延ディスパッチと呼んでいた

システム状態管理機能

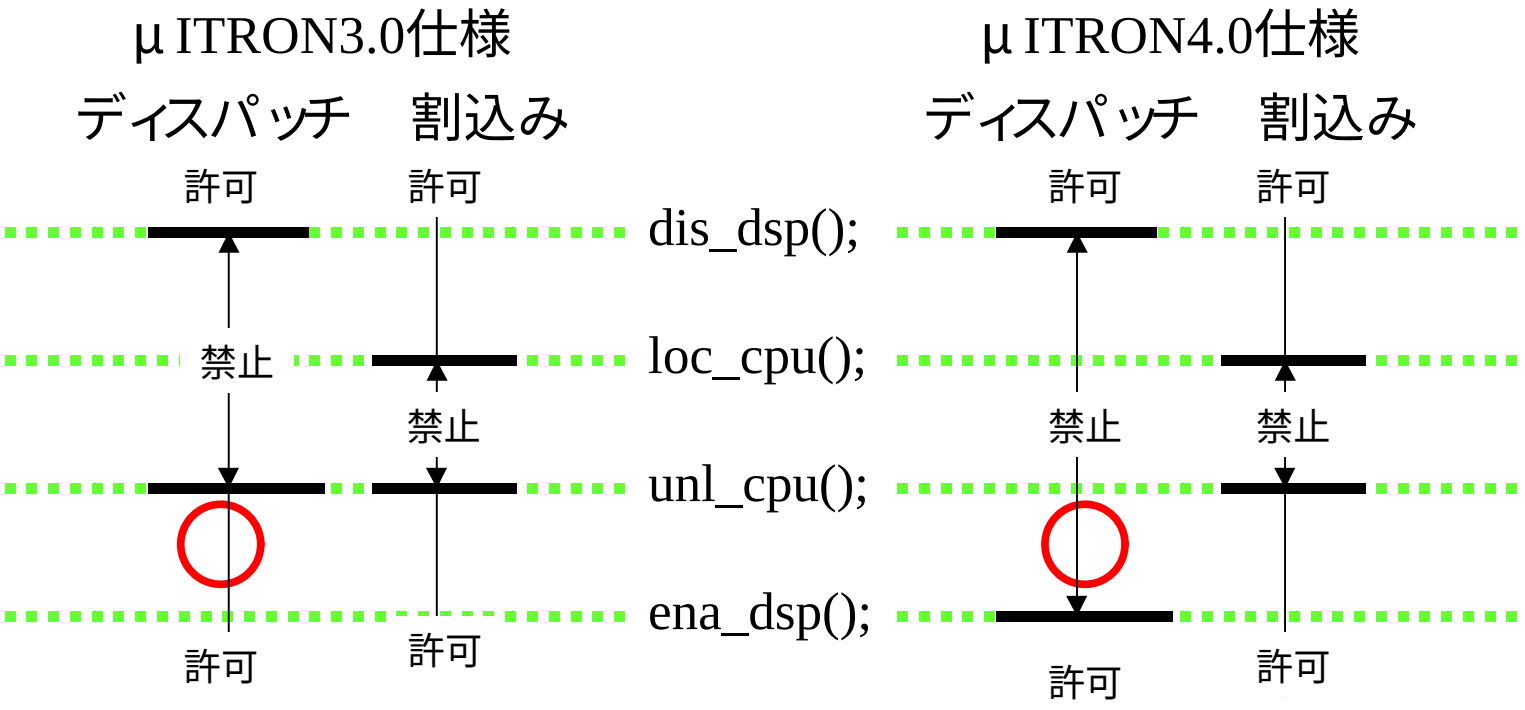
サービスコール名	機能
loc_cpu iloc_cpu	CPUロック状態への移行【S】
unl_cpu iunl_cpu	CPUロック状態の解除【S】
dis_dsp	ディスパッチの禁止【S】
ena_dsp	ディスパッチの許可【S】
sns_ctx	コンテキストの参照【S】
sns_loc	CPUロック状態の参照【S】
sns_dsp	ディスパッチ禁止状態の参照【S】
sns_dpn	ディスパッチ保留状態の参照【S】
ref_sys	システムの状態の参照

CPUロック状態

- 割込みハンドラ、タイムイベントハンドラ、ディスパッチャは起動されない
- CPUロック状態で呼び出すことのできるサービスコール
  loc_cpu/iloc_cpu,unl_cpu/iunl_cpu
  sns_ctx,sns_loc,sns_dsp,sns_dpn,sns_tex
- 割込みハンドラ起動直後は実装依存
- 割込みサービスルーチン、タイムイベントハンドラの起動直後と終了直前はCPUロック解除状態
  CPUロック解除！＝割込み許可
  上記サービスコール以外のサービスコールを使用可能

ディスパッチ禁止状態

- 待ちになる可能性のあるサービスコールは発行できない
- CPUロックとディスパッチ禁止は独立



排他制御処理の実現方法

- 例）割込み禁止にする必要がある処理

```
cpu_locked = sns_loc();

if( !cpu_locked )
        loc_cpu();
割込み禁止で行うべき処理
if( !cpu_locked )
        unl_cpu();
```

少ないオーバーヘッドでシステムの状態を知ることができるAPI（sns_yyy）

μITRON3.0仕様ではこの処理の前にディスパッチ禁止であった場合でも、最後のunl_cpuの処理でディスパッチ許可になる

ディスパッチ保留状態

sns_dpn　(sense dispatch pending)

ディスパッチ保留状態の参照

- ディスパッチャよりも優先順位の高い処理が実行されている間
- CPUロック状態
- ディスパッチ禁止状態

サービスコールとコンテキスト

- 非タスク専用のサービスコール
  iact_tsk,iwup_tsk,irel_wai,iras_tex,isig_sem,iset_flg,ipsnd_dtq,
  ifsnd_dtq,isig_tim,irot_rdq,iget_tid,iloc_cpu,iunl_cpu
- どのコンテキストからも呼び出せるサービスコール
  sns_ctx,sns_loc,sns_dsp,sns_dpn,sns_tex
- タスクコンテキスト専用のサービスコール
  上記以外のすべてのサービスコール

サービスコールを間違ったコンテキストで使用した場合は未定義
→　エラーを返すならばE_CTX

非タスクコンテキストから発行可能なサービスコール一覧

サービスコール名	機能
iact_tsk	タスクの起動【】
iwup_tsk	タスクの起床【】
irel_wai	待ち状態の強制解除【】
iras_tex	タスク例外処理の要求【】
isig_sem	セマフォ資源の返却【】
iset_flg	イベントフラグのセット【】
ipsnd_dtq	データキューへの送信（ポーリング）【】
ifsnd_dtq	データキューへの強制送信【】
isig_tim	タイムティックの供給【】
irot_rdq	タスクの優先順位の回転【】
iget_tid	実行状態のタスクIDの参照【】
iloc_cpu	CPUロック状態への移行【】
iunl_cpu	CPUロック状態の解除【】

非タスクコンテキストからのサービスコール

μITRON3.0仕様との違い

- サービスコール名が必ず“i”で始まらなければならない
  割込みハンドラなどからの呼び出し方法の統一
- 非タスクコンテキストからのサービスコールを限定
  規定されているもの以外はμITRON4.0仕様外の独自拡張

タスクへイベントの発生を通知するもの

サービスコールの追加

- 非タスクコンテキストから呼び出せるサービスコールを実装独自に追加することができる
- 追加したサービスコールは先頭に“i”を付けるだけでよい。vをつける必要はない
- 独自に拡張したサービスコールでも非タスクコンテキストから呼べるものは先頭に“i”をつける

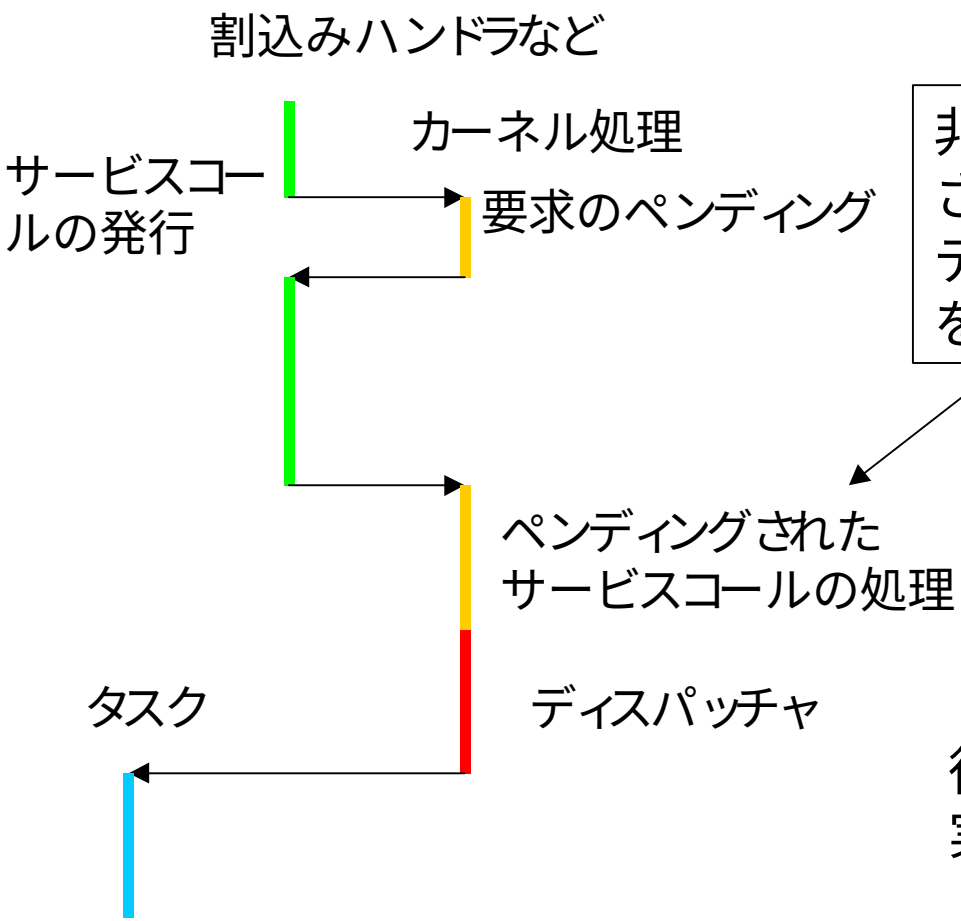
例：

clr_flgを非タスクコンテキストから呼べるように拡張した場合→iclr_flg

clr_flgに実装独自の仕様に変更した場合→vclr_flg

vclr_flgを非タスクコンテキストから呼べるように拡張した場合→ivclr_flg

サービスコールの遅延実行 (1)



非タスクコンテキストから呼び出されたサービスコールの実行はディスパッチャの起動までに実行を遅延することを許す

従来は仕様では規定していないが実装独自に行っていたものがある

サービスコールの遅延実行（2）

- 遅延実行できないサービスコール
  iget_tid,iloc_cpu,iunl_cpu,
  sns_ctx,sns_loc,sns_dsp,sns_dpn,sns_tex
  これらのサービスコールは遅延実行すると意味がなくなる
- 遅延実行したときのエラー
  即時に実行したときに発生すべきエラーが返せない可能性がある
  (例)iwup_taskでの起床要求カウントのオーバーフローなど

サービスコールを発行した時点ではE_OKを返す
要求をキューイングするメモリ領域が無い場合はE_NOMEMを返す

拡張サービスコールルーチン

μITRON3.0仕様の拡張SVCハンドラに対応

μITRON3.0仕様

定義

def_svc（機能コード，　定義情報　）；

呼出し

レジスタに機能コードや引数をセット

通常のサービスコール呼出し（ソフトウェア割込みなど）

サービスコール管理機能

サービスコール名	機能
DEF_SVC	拡張サービスコールの定義(静的API)
def_svc	拡張サービスコールの定義
cal_svc	サービスコールの呼出し

μITRON4.0仕様

定義

def_svc（機能コード, 定義情報 ）;

呼出し

cal_svc (機能コード, 引数. . . . .);

このサービスコールで標準のサービスコールを呼び出すことができるかどうかは実装定義

これ自身は機能コードがないので、呼び出すことはできない