組込みリアルタイムOS向けテストツールのマルチプロセッサ拡張

2011年1月25日

株式会社デジタルクラフト
金 ハンソル

アジェンダ

1. はじめに
2. シングルプロセッサ向けのテストツール
3. マルチプロセッサ向けのテストケース
4. テストツールのマルチプロセッサ拡張
5. 実施結果と評価
6. まとめ

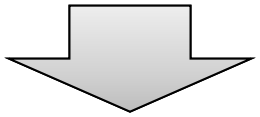
背景

組込みシステムにおいてマルチプロセッサの利用が増加

低消費電力で動作し性能を向上させる方法として，シングルプロセッサでクロック周波数を上げるよりも，マルチプロセッサを用いて並列処理を可能にする方が有利

マルチプロセッサ向けRTOSのテスト手法の不在

マルチプロセッサに対応したRTOS に関しては，そもそもテスト経験が少ないために，テスト手法が確立されていない



マルチプロセッサ向けRTOSのテスト手法開発へ

コンソーシアム型共同研究

※　2009年度は，シングルプロセッサ向けRTOSのAPIに着目したテスト（APIテスト）を開発

シングルプロセッサ向けRTOSのAPIテストの開発経験を活かし，
マルチプロセッサ向けRTOSを対象にして，APIテストを実施

TOPPERS新世代カーネル

- μITRON仕様をベースとして，信頼性，安全性，ソフトウェアポータビリティを向上させるための改良・拡張
- 新世代カーネルの仕様は“TOPPERS 新世代カーネル統合仕様書”にまとめられている
  - シングルプロセッサ向けRTOS：TOPPERS/ASPカーネル
  - マルチプロセッサ向けRTOS　：TOPPERS/FMPカーネル

ASP Kernel
Advanced Standard Profile Kernel

FMP Kernel
Flexible MultiProcessor Kernel

今年度のAPIテスト対象のカーネル

TOPPERSプロジェクトとは？

- TOPPERSカーネルを開発/公開しているNPO法人
- ITRON仕様のRTOS，ミドルウェア等を開発
- オープンソースで公開
- 実製品にも利用されている

TOPPERS
Toyohashi OPen Platform for Embedded Real-time Systems

http://www.toppers.jp/

アジェンダ

1. はじめに
2. シングルプロセッサ向けのテストツール
3. マルチプロセッサ向けのテストケース
4. テストツールのマルチプロセッサ拡張
5. 実施結果と評価
6. まとめ

APIテストの概要

APIテストとは？

- カーネルが提供しているAPIが統合仕様書に定められている通りに正しく振舞うことを確認するテスト
- APIはシステムの状態を変更することが可能

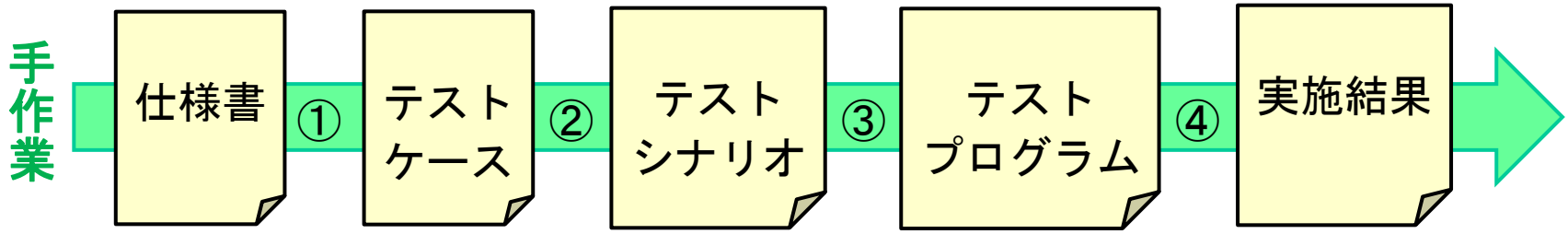
APIテストの流れ

①統合仕様書からテスト対象であるAPI の仕様と振舞いを理解し，テストケースを抽出

②テストケース毎にテストシナリオを作成

③テストシナリオを実現するためのテストプログラムを開発

④テストプログラム実行



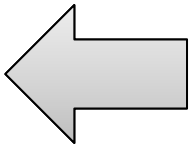
テストケースとテストシナリオ

対象タスクが休止状態である場合には，対象タスクに対してタスク起動時に行うべき初期化処理が行われ，対象タスクは実行できる状態になること

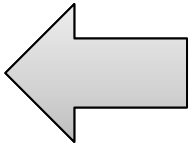
テストケースの例

前状態
　　優先度：中のタスク1が実行状態
　　優先度：高のタスク2が休止状態
処理
　　タスク1がact_tsk(タスク2)を発行し，
　　エラーコードとしてE_OKが返る
後状態
　　タスク1が実行可能状態
　　タスク2が実行状態

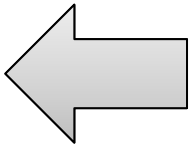
テストシナリオの例



テストを行うAPIを呼ぶ時点でのシステムの状態



テストを行うAPIとその引数，戻り値



APIを呼んだ後のシステムの状態

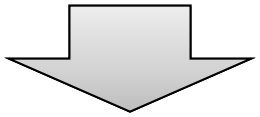
※ タスク＝スレッド

ハンドコーディングでのテストプログラム開発の問題点

1. テストプログラムの可読性，保守性の低下
   テストケースの実現方法が複数存在するため，開発担当者が異なることによるばらつき発生

2. テストの開発工数
   ASPカーネルのAPIは121個存在し，抽出したテストケースは1,669件

3. 異なるRTOSへの流用不可
   同一テストシナリオに対するRTOS毎の重複開発



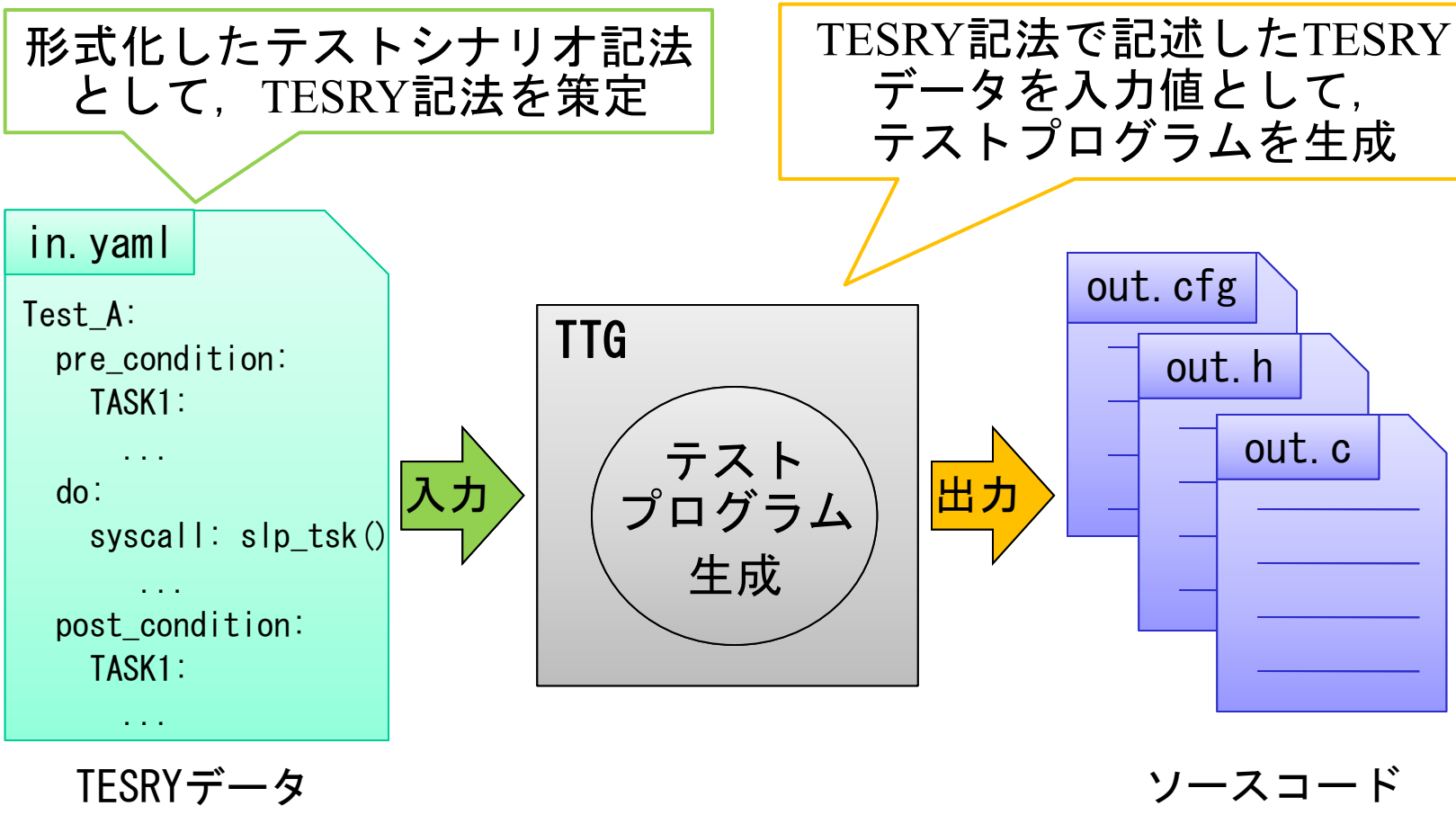
テストプログラム作成をするツールの開発へ

TOPPERS Test Generator（TTG）

テストプログラムを生成するツール

形式化したテストシナリオ記法として，TESRY記法を策定

TESRY記法で記述したTESRYデータを入力値として，テストプログラムを生成



TESRYデータ

ソースコード

TEst Scenario for Rtos by Yaml(TESRY)記法

テストシナリオをYAML形式で記述するもの

前状態・後状態の指定

前状態を"pre_condition"内に,
後状態を "post_condition"内に記述

処理の指定

発行するAPIとその引数と戻り値を
"do"内に記述

TTGは，テスト対象のAPIを
実行し，後状態をチェックする
プログラムを生成

```
pre_condition:
  TASK1:
    type    : TASK
    tskpri  : TSK_PRI_MID
    tskstat: running
  TASK2:
    type    : TASK
    tskpri  : TSK_PRI_HIGH
    tskstat: dormant
do:
  id      : TASK1
  syscall: act_tsk(TASK2)
  ercd    : E_OK
post_condition:
  TASK1:
    tskstat: ready
  TASK2:
    tskstat: running
```

前状態

処理

後状態

※YAML：造化データやオブジェクトを文字列に直列化するためのデータ形式

TESRY記法の時刻の指定

指定した時刻でのシステム状態をチェックするための記述

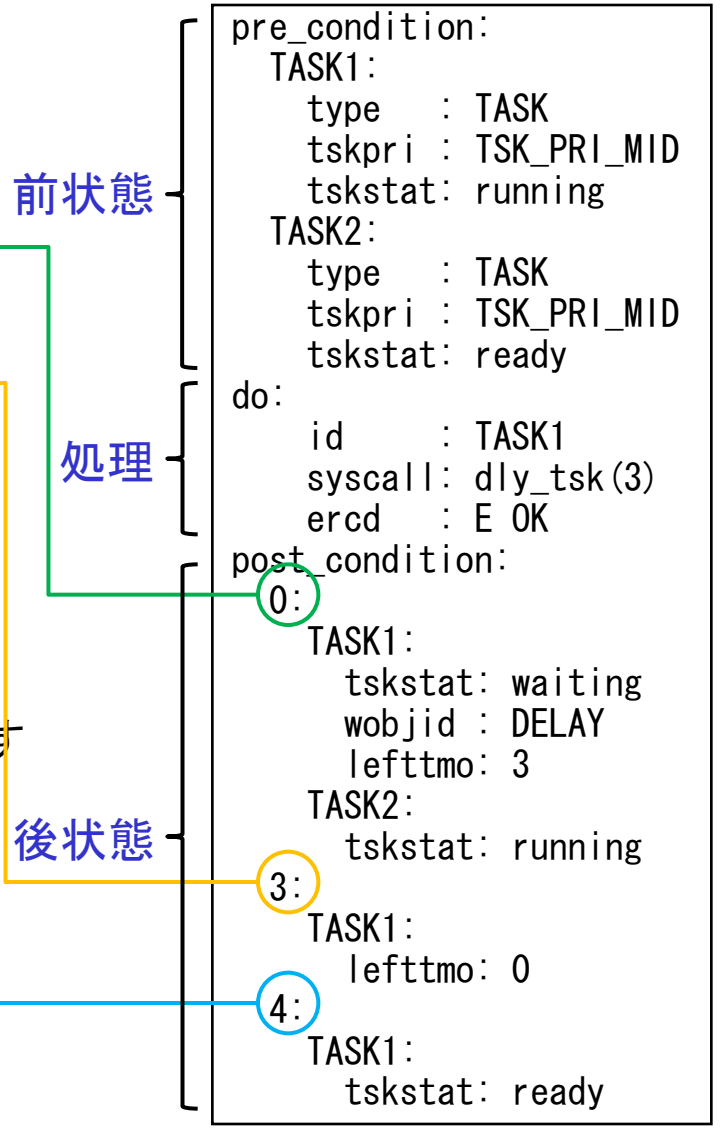
“0” : dly_tskによりTASK1は待ち状態となる

“3” : 3ms後まではタスクの状態は変化なし

“4” : 4ms後にTASK1は待ち解除され，実行状態になる

※ “3”，“4”は，結果を確認する時刻（ミリ秒）を示す

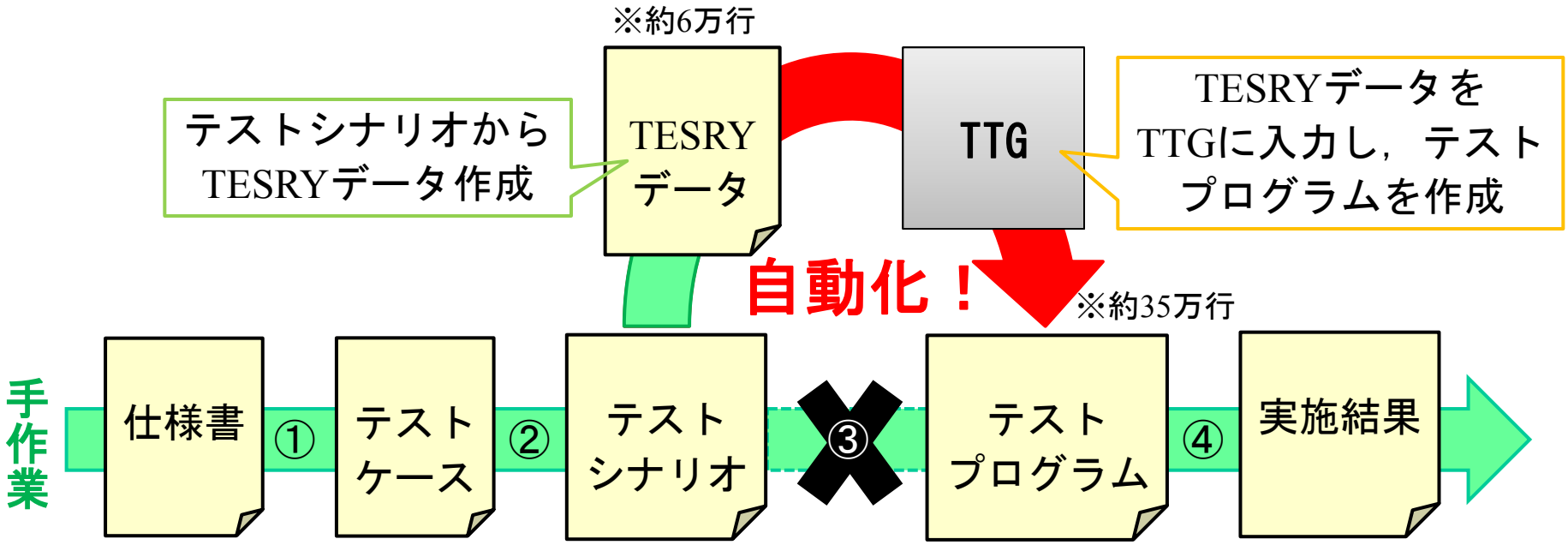
TTGは，時間制御関数を用いて，RTOSのシステム時刻を制御するプログラムを生成



TTGの導入効果

テストプログラムの保守性と再利用性の向上

開発規模を約1/6に圧縮



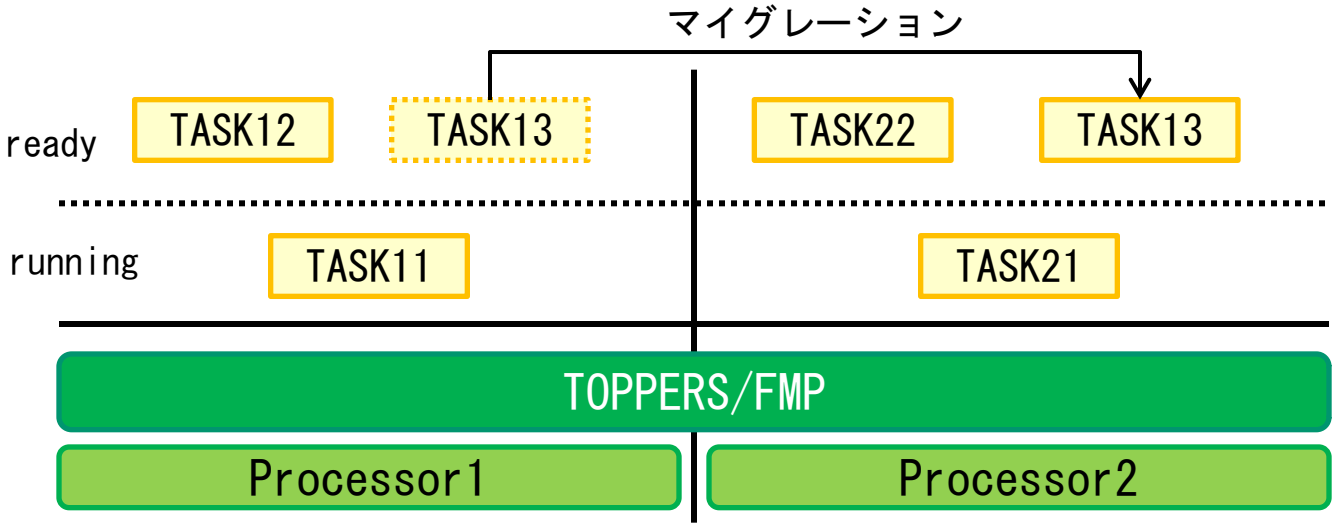
アジェンダ

1. はじめに
2. シングルプロセッサ向けのテストツール
3. マルチプロセッサ向けのテストケース
4. テストツールのマルチプロセッサ拡張
5. 実施結果と評価
6. まとめ

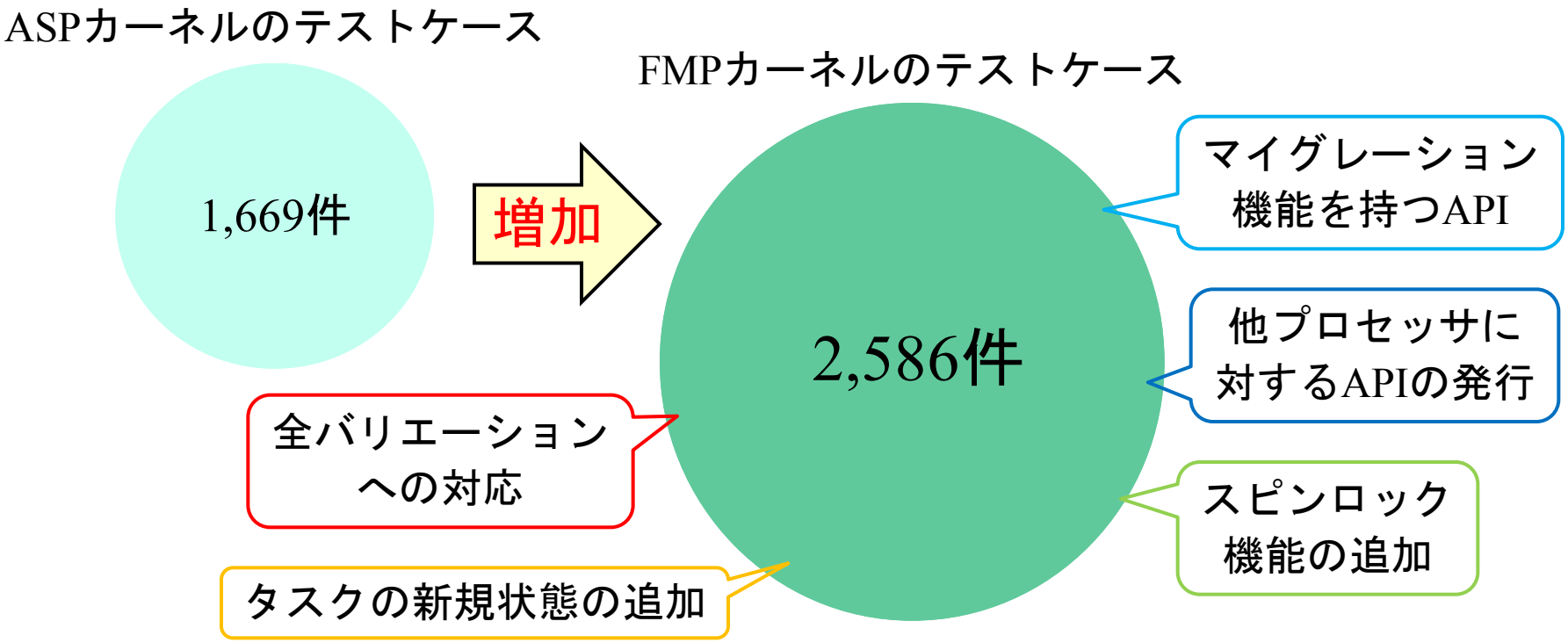
TOPPERS/FMPカーネル

ASPカーネルを，マルチプロセッサ向けに機能拡張したRTOS

- 複数のタスクが同時に動く
- タスクは割付けられたプロセッサ上で動く
- タスクの割付けプロセッサをAPIで変えられる（マイグレーション）
- 他プロセッサに割付けられているタスクに対してAPIの発行
- プロセッサ間の同期機構（スピンロック）
- 多様なハードウェアアーキテクチャに対応



FMPカーネルのテストケース



既存のテストツールは，FMPカーネルで追加された仕様に対応できていない

アジェンダ

1. はじめに
2. シングルプロセッサ向けのテストツール
3. マルチプロセッサ向けのテストケース
4. テストツールのマルチプロセッサ拡張
   - ① FMPカーネルで新設された要素に対する対応
   - ② プロセッサ間の同期処理実現
   - ③ プロセッサ間の時間制御の実現
   - ④ バリエーションへの対応
5. 実施結果と評価
6. まとめ

①FMPカーネルで新設された要素に対する対応

- **割付けプロセッサ（prcid）**
  TTGが割付けプロセッサの指定を受け付け，処理単位を割付けさせるプログラムを生成

- **タスクの新規状態（running-waitspin）**
  スピンロックを取得待ち状態の指定を受け付け，処理単位をスピンロック取得待ち状態にするプログラムを生成

- **スピンロック（SPINLOCK）**
  procidにおいて指定された処理単位に，スピンロックを取得させるプログラムを生成

※処理単位：カーネルが実行制御の対象としているプログラムのこと（タスク、割込みなど）

```
pre_condition:
  TASK1:
    type    : TASK
    tskstat: running
    prcid   : 1
  TASK2:
    type    : TASK
    tskstat: running
    prcid   : 2
  SPN1:
    type    : SPINLOCK
    spnstat: TSPN_LOC
    procid : TASK1
  CPU_STATE1:
    type    : CPU_STATE
    loc_cpu: true
    prcid   : 1
do:
  id      : TASK2
  syscall: loc_spn(SPN1)
  ercd    : E_OK
post_condition:
  TASK2:
    tskstat: running-waitspin
    spinid : SPN1
```

②プロセッサ間の同期処理

前状態，処理，後状態の定義

TESRY記法の前状態，処理，後状態の記述にて，全プロセッサの状態を指定できるように拡張

次のフェーズに進む瞬間に，必ず全プロセッサのシステムの状態がTESRYデータに記述されている通りになっている必要がある

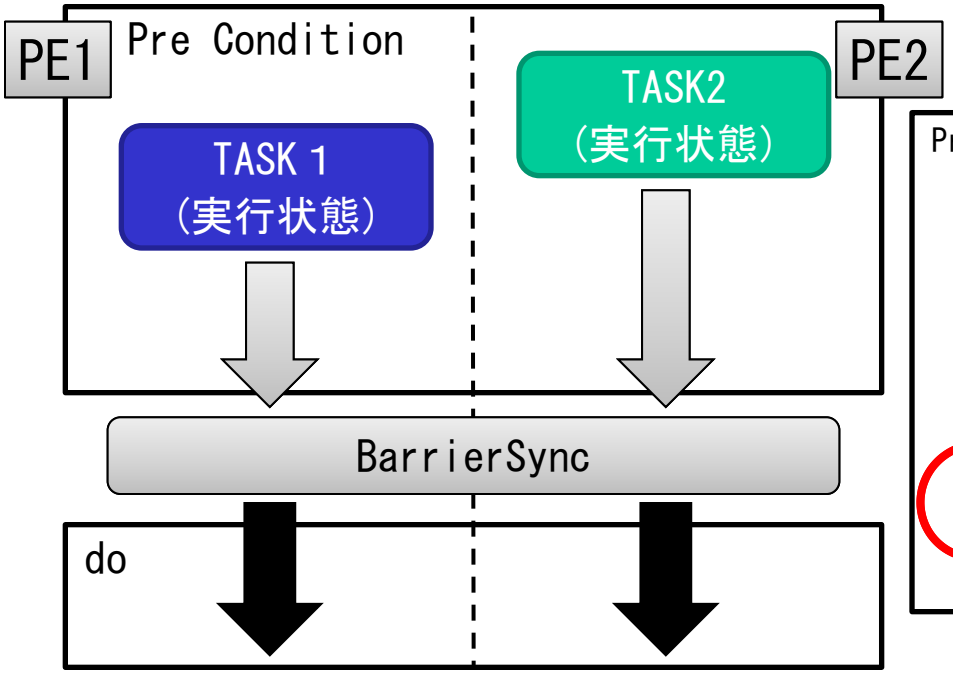
```
pre_condition:
  TASK1:
    type    : TASK
    tskstat: running
    prcid   : 1
  TASK2:
    type    : TASK
    tskstat: running
    prcid   : 2
do:
  id       : TASK1
  syscall: sus_tsk(TASK2)
  ercd     : E_OK
post_condition:
  TASK2:
    tskstat: suspended
```

複数のプロセッサ間で，
処理単位を実行するタイミングを
合わせることが必要

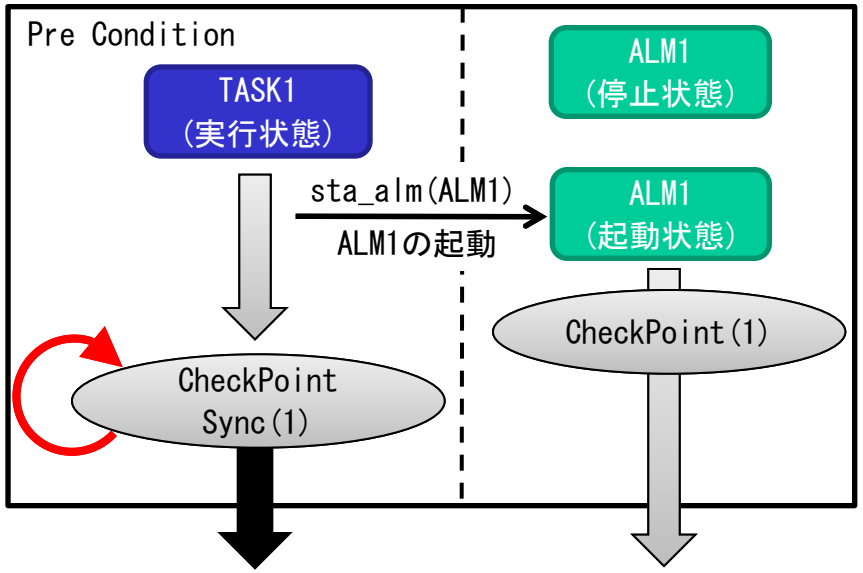
②プロセッサ間の同期処理の実現

TTGが入力されたTESRYデータに応じて，適切な
同期処理を行うテストプログラムを生成する機能を追加

① 指定したプロセッサ間での実行タイミングを合わせる同期（BarrierSync）



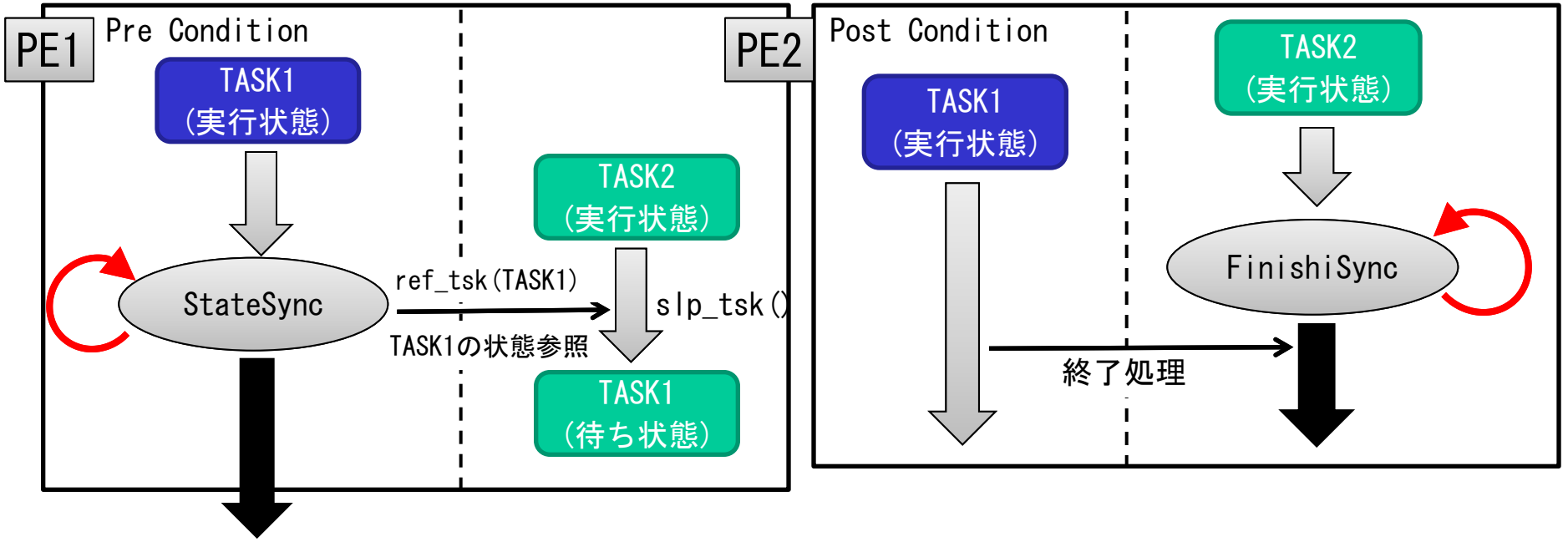
② 他プロセッサの処理単位の特定の処理終了を待つ同期
（CheckPointSync）



②プロセッサ間の同期処理の実現

③ 特定の処理単位が特定の状態になるまで待つ同期(StateSync)

④ 実行中の処理単位を終了させないための同期(FinishSync)



これらの同期機構を用いることにより,
テストプログラムのプロセッサ間の同期処理を実現

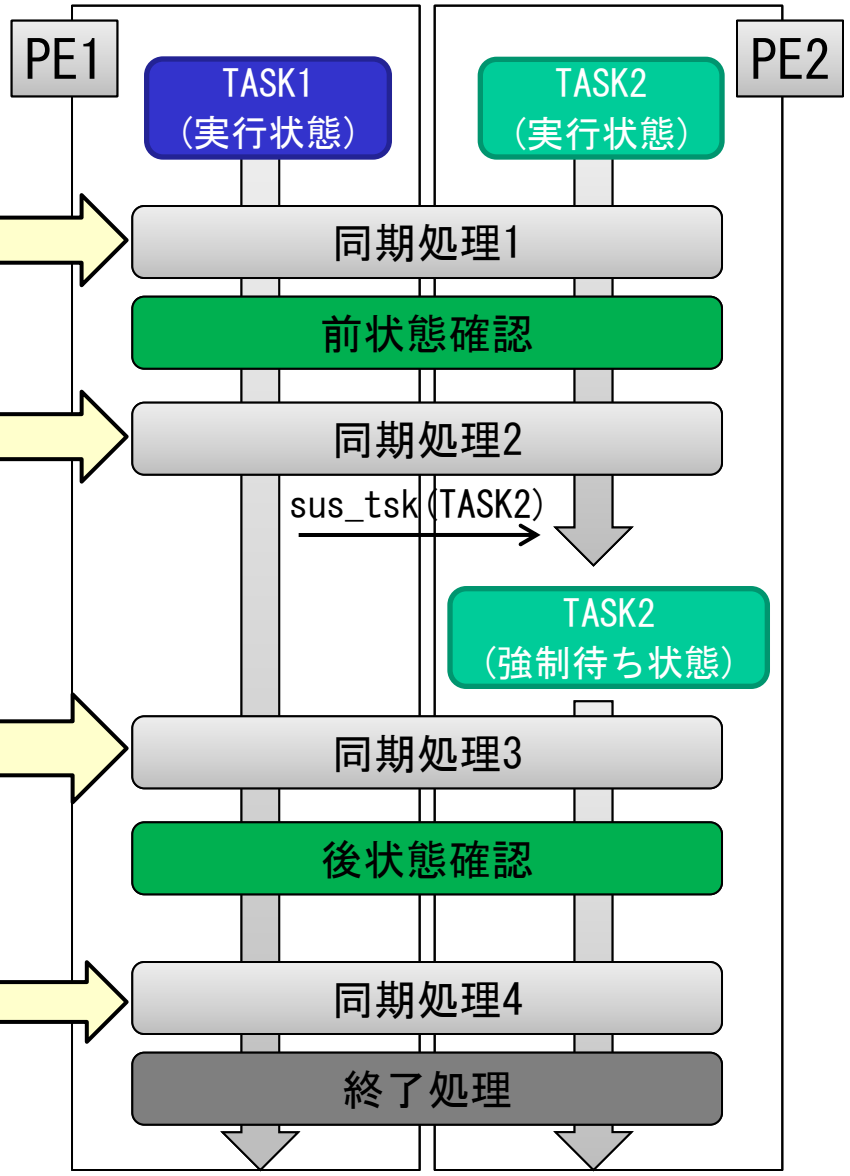
② TTGが生成する同期処理

全プロセッサの**前状態作成**が
完了するまで待つための同期
（CheckPointSync， StateSync ）

全プロセッサの**前状態確認**を
終えるまで待つための同期
（ BarrierSync ）

テスト対象のAPIの発行処理が終わるまで，
他のプロセッサでの処理を，後状態へと**先に進めさせない**ための同期
（ BarrierSync， StateSync ）

後状態確認の**処理が終わる**まで，実行状態の
処理単位を終了させないための同期
（ FinishSync ）



③プロセッサのシステム時刻制御

時刻の定義

TESRY記法の処理，後状態の記述において，全プロセッサのシステム状態を確認する時刻を指定できるように拡張

全プロセッサの時刻を揃えるためには，各プロセッサのシステム時刻を制御することが必要

次のフェーズに進む時に，必ず全プロセッサのシステムの状態がTESRYデータに記述されている通りになっている必要がある

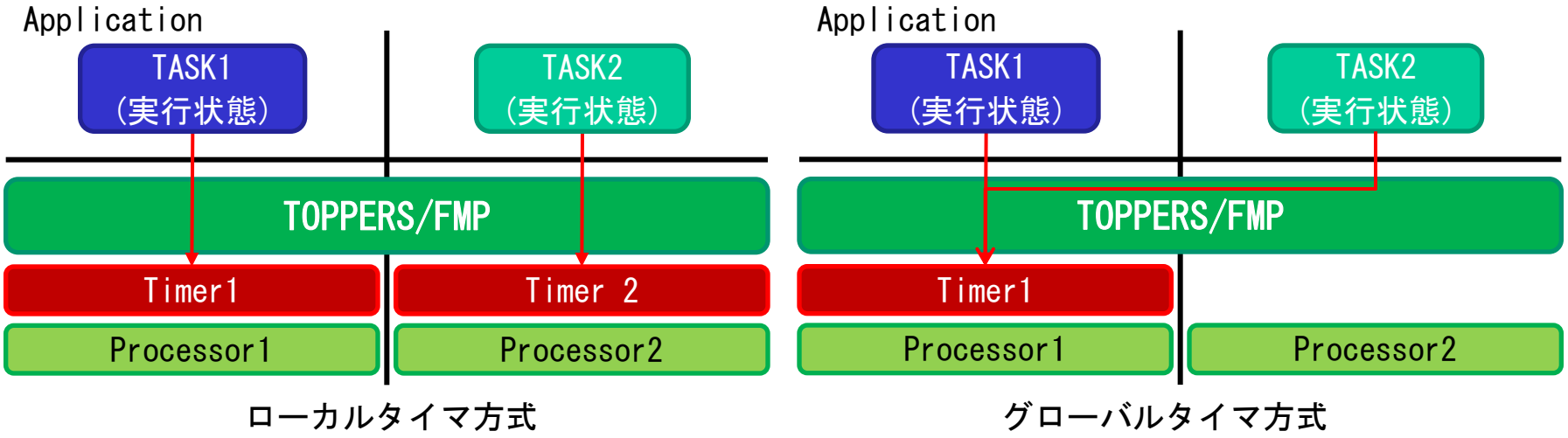
```
variation:
  timer_arch: local
pre_condition:
  TASK1:
    type    : TASK
    tskstat: running
    prcid   : 1
  ALM1:
    type    : ALARM
    almstat: TALM_STP
    hdlstat: STP
    prcid   : 1
do:
  id      : TASK1
  syscall: msta_alm(ALM1,3,2)
  ercd    : E_OK
post_condition:
  0:
    ALM1:
      almstat: TALM_STA
      hdlstat: STP
      lefttim: 3
      prcid   : 2
  3:
    ALM1:
      lefttim: 0
  4:
    ALM1:
      almstat: TALM_STP
      hdlstat: ACTIVATE
```

③プロセッサのシステム時刻制御の問題点

タイマ方式（OSのシステム時刻管理方式）

- ローカルタイマ方式　：プロセッサ毎にシステム時刻が独立
- グローバルタイマ方式：全プロセッサで1つのシステム時刻を管理

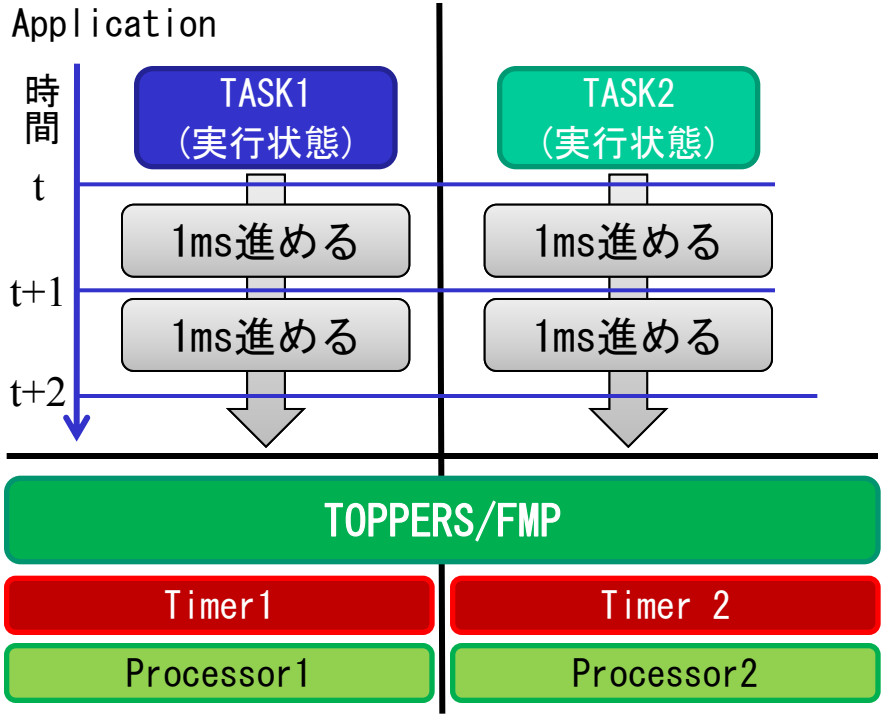
タイマ方式毎に，テストにおける
システム時刻の進め方を変えることが必要



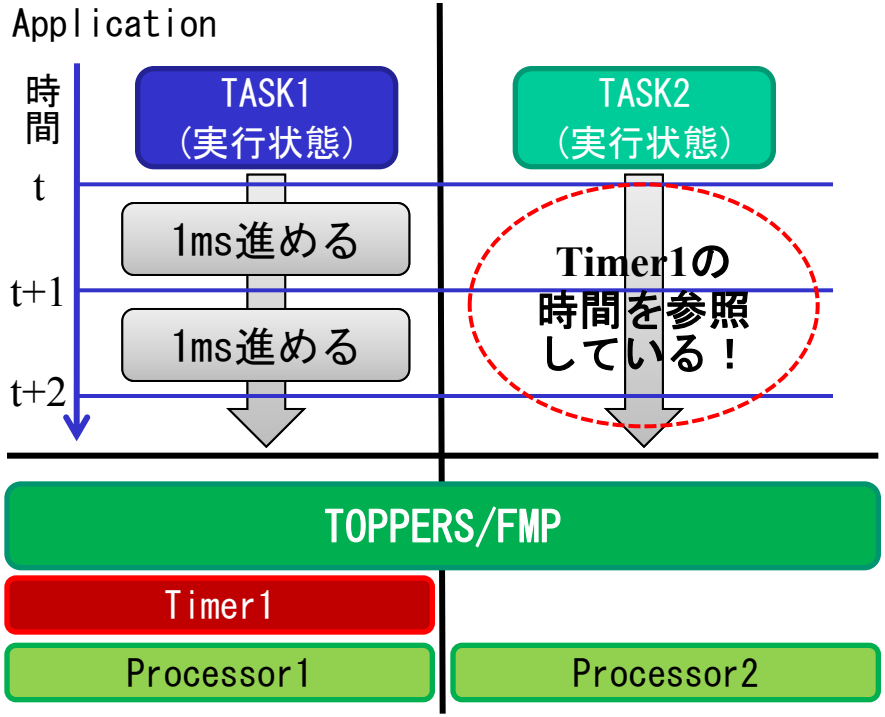
③プロセッサのシステム時刻制御の実現

プロセッサ毎にシステム時刻を進められるように拡張

ローカルタイマ方式

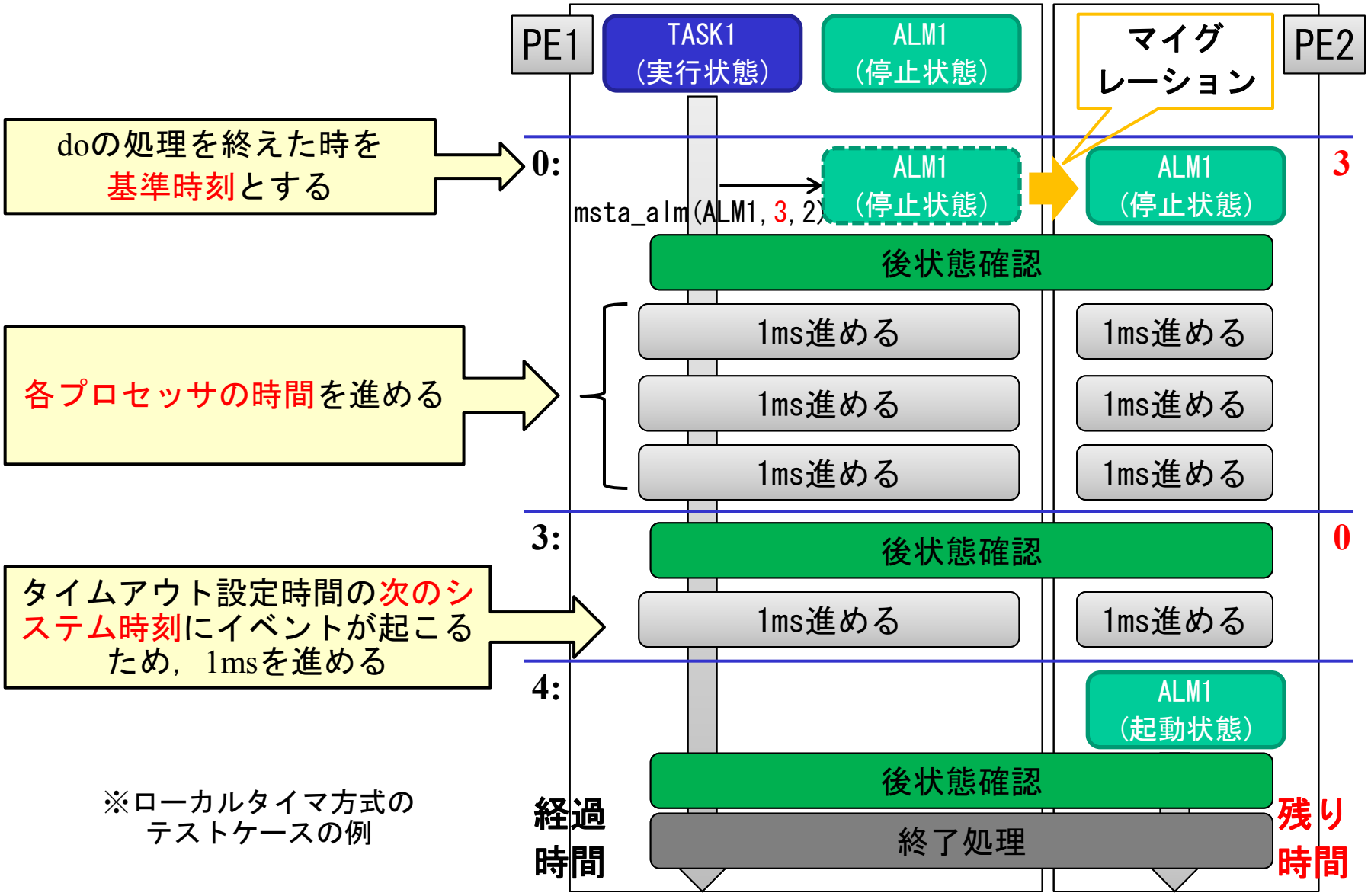


グローバルタイマ方式



※ASPと同じ時間の進め方で問題ない

③ TTGが生成するシステム時刻制御



※ローカルタイマ方式のテストケースの例

④バリエーションへの対応

バリエーションとは？

マルチプロセッサシステムのハードウェアアーキテクチャの違い

・プロセッサの種類や数　・メモリとタイマ　・プロセッサ間排他制御方法

テストケースのバリエーション判別

FMPカーネルでは，バリエーションによっては実施できないテストケースが多数存在するため，実行可能なテストケースを取捨選択することが必要

➡ テストケースは数千件に及ぶため，ユーザに実行できるテストケースを取捨選択させるのは非効率的

TTGに実行可能なテストケースを取捨選択させる機能を設けることで，実施容易性を確保

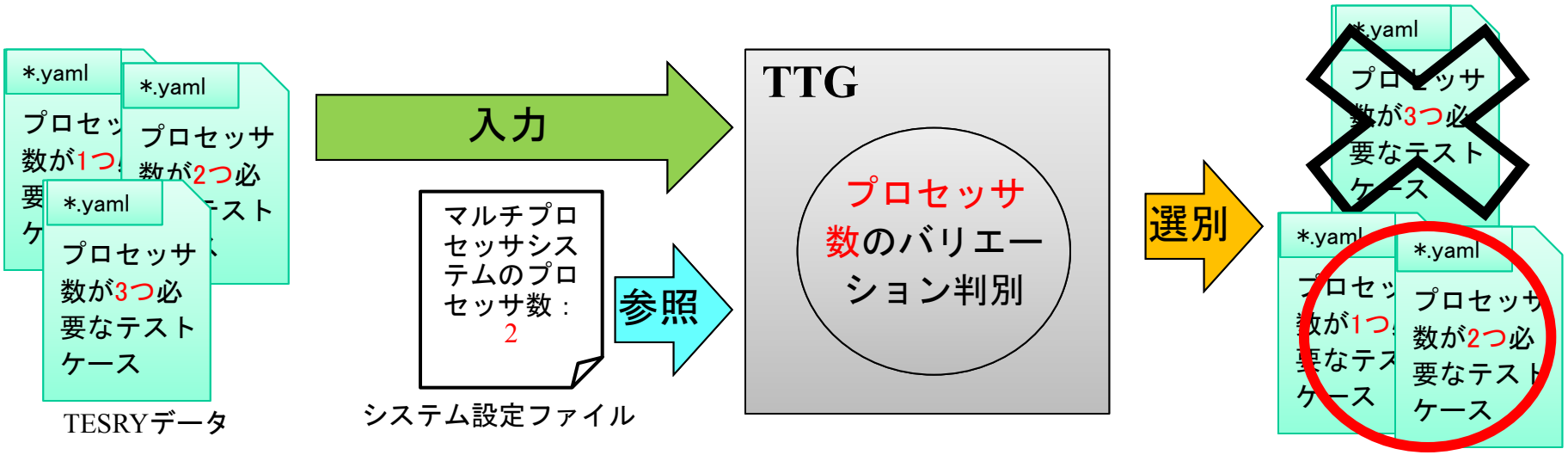
※本発表では，プロセッサ数とタイマ方式のみ説明

④ー①プロセッサ数の取捨選択

テストに必要なプロセッサ数

テストプログラムに必要なプロセッサ数より，テストが限定されることがある

➡ テストが実行可能なTESRYデータかどうかを判断



④－②タイマ方式の取捨選択

実行結果が異なるAPI

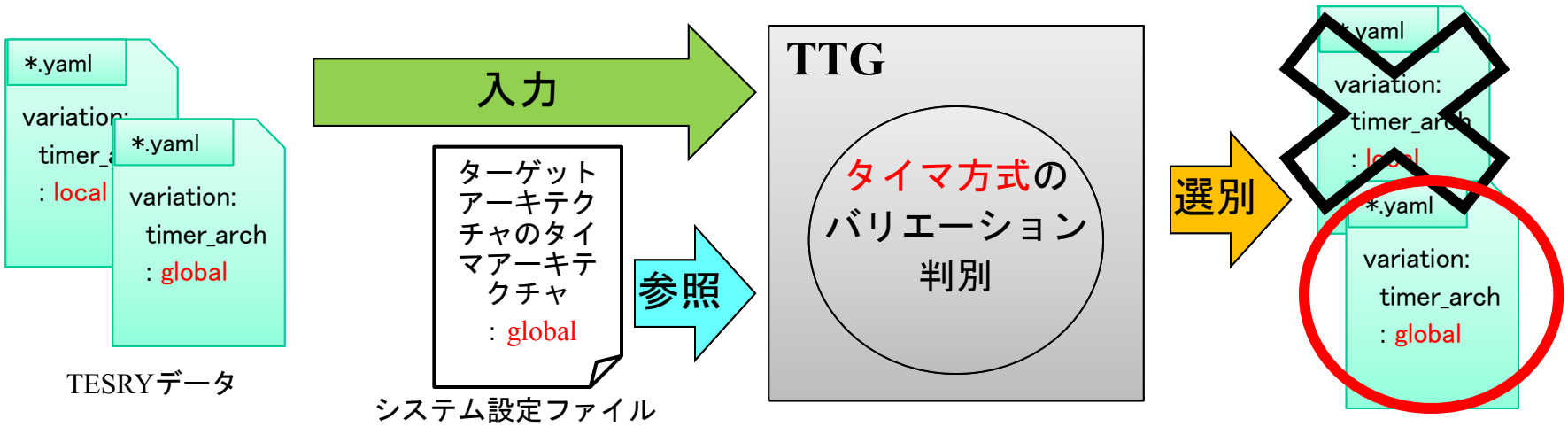
タイマ方式によっては，テストケースの実行結果がAPIにより変わる場合が存在

➡ テスト実行時には，タイマ方式に応じて実行可能なテストケースのみを抽出することが必要

```
variation:
  timer_arch: local
pre_condition:
  TASK1:
    type    : TASK
    tskstat: running
    prcid   : 1
    ...
```

TESRY記法のバリエーション対応

新たにTESRY記法へ“variation”属性を設け，ローカルタイマ方式とグローバルタイマ方式で実行できるテストケースを選別



アジェンダ

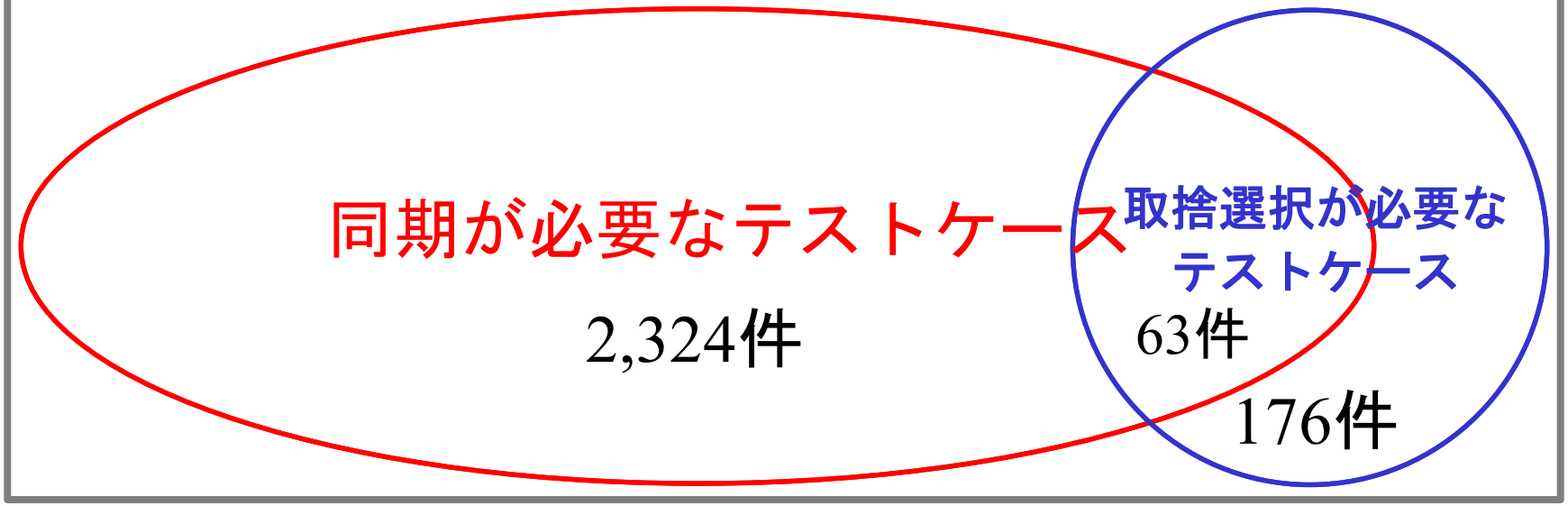
1. はじめに
2. シングルプロセッサ向けのテストツール
3. マルチプロセッサ向けのテストケース
4. テストツールのマルチプロセッサ拡張
5. 実施結果と評価
6. まとめ

実施結果1

FMPカーネルの全テストケースの
テストプログラムを生成することが可能

マルチプロセッサのハードウェアアーキテクチャの違いに対し、
柔軟に対応することが可能

全テストケース：2,586件



実施結果2

FMPカーネルの不具合を27件検出

- 冗長した分岐
- 仕様と実装の違い

  ※タスクを割付けることができる，割付け可能プロセッサのチェックが漏れていた
- 意図した状態に遷移しない

  ※過渡状態のタスクが自タスクを終了するAPIを呼び出した場合に，強制待ち状態となることがある

評価

マルチプロセッサ拡張

目的としているAPIに対して，適切な同期を設けて
意図したテストを実施することができた

バリエーション対応

TTGが実行可能なテストケースを取捨選択する
ことにより，工数を削減することができた

※ TTG自体のテストも実施

まとめ

- 名古屋大学を中心としたFMPカーネルの検証に関するコンソーシアム型研究組織は，2009年度に開発したASPカーネルのAPIテストスイートとTTGを拡張する形で，2010年度，FMPカーネルのAPIテストスイートを開発し，APIテストを実施した

- FMPカーネルに追加された仕様の対応，プロセッサ間の同期機構・システム時刻制御機能の追加，バリエーション対応により，FMPカーネルに対するAPIテストを実施することが可能となった

- FMPカーネルの不具合を27件検出した