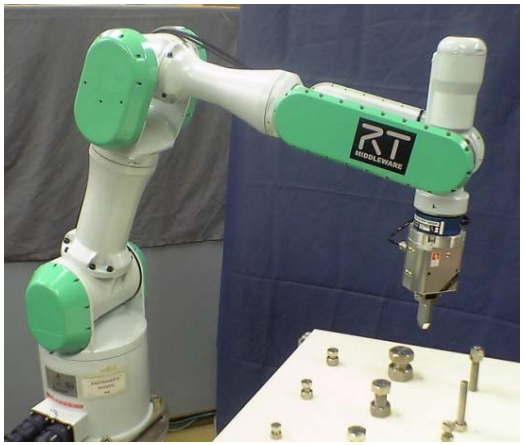
RTミドルウェアによる実時間ロボット制御系の構築とソフトウェア教育

静岡大学
大学院工学研究科
機械工学専攻
清水 昌幸

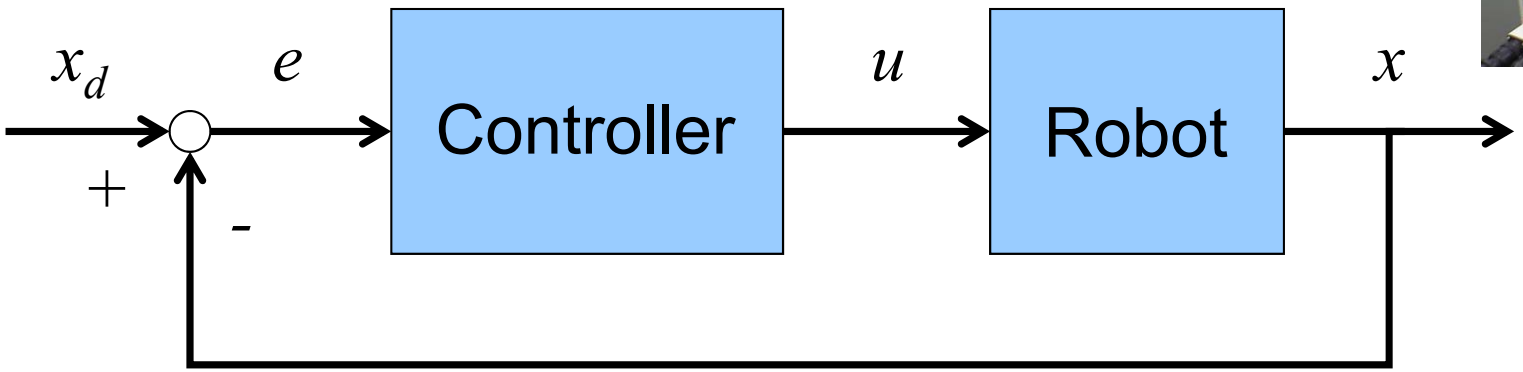
内容

- RTミドルウェアを用いた実時間ロボット制御系の構築
  - 実時間制御の実現方法.
  - ART-Linuxを用いた実現例と実時間性能.
- ソフトウェア教育
  - 研究室配属学生（学部4年生）への導入教育の実施内容と結果.

ロボット制御系の一例

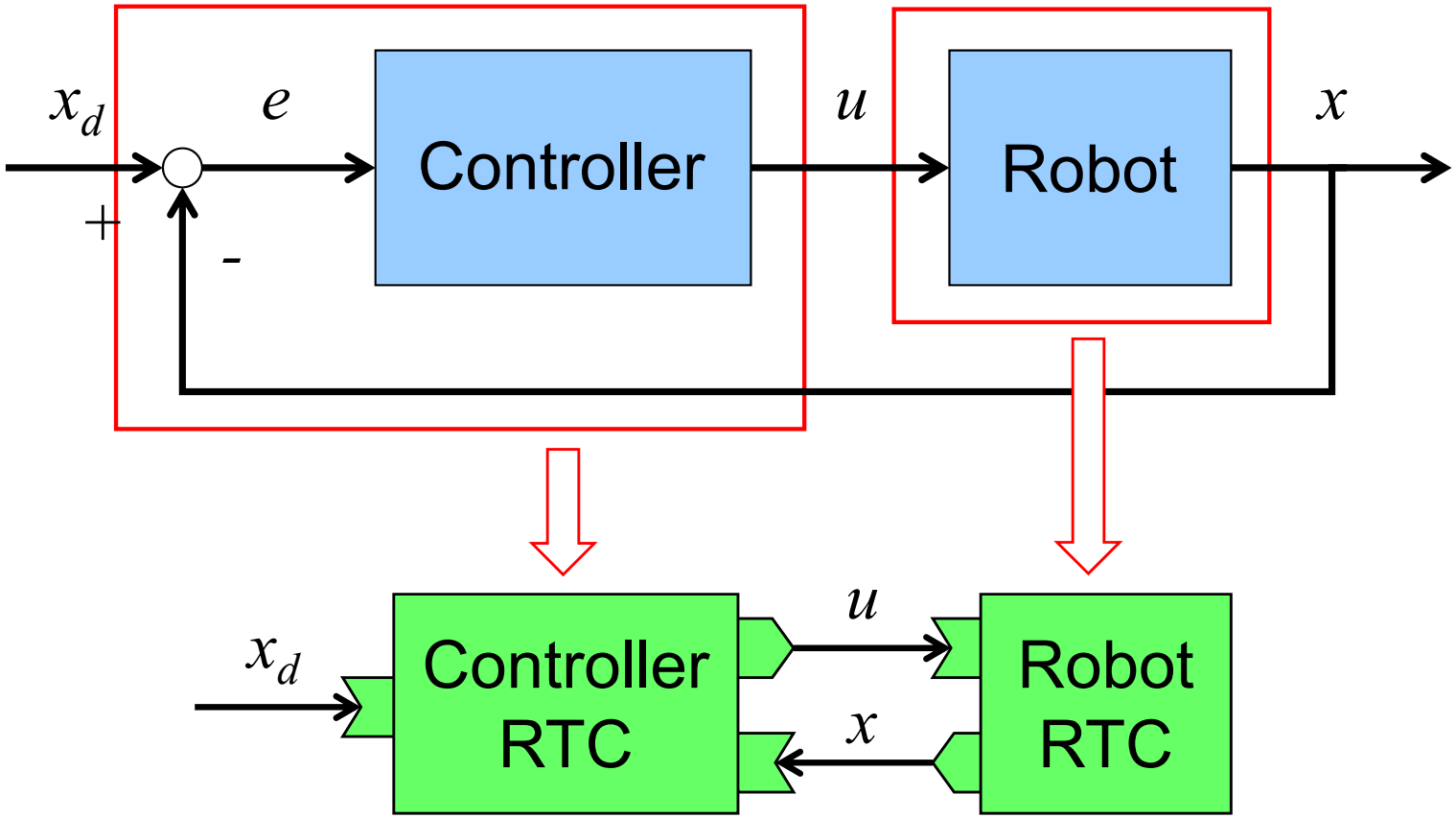


- フィードバック制御系



x：ロボットの現在状態
x_d：ロボットの目標状態
e：偏差
u：操作量

RTコンポーネントを用いた制御系の実現



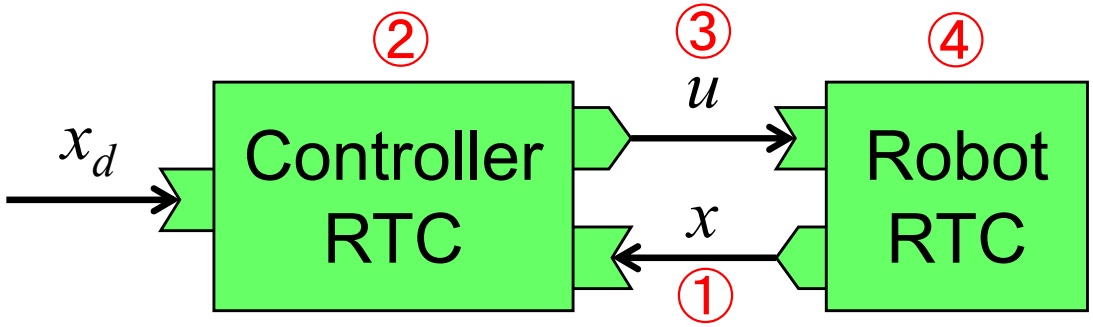
ロボットRTC：操作量uに従ってロボットを制御し，現在状態xを出力する．

制御器RTC：現在状態xと目標状態x_dに基づいて計算した操作量uを出力する．

制御処理

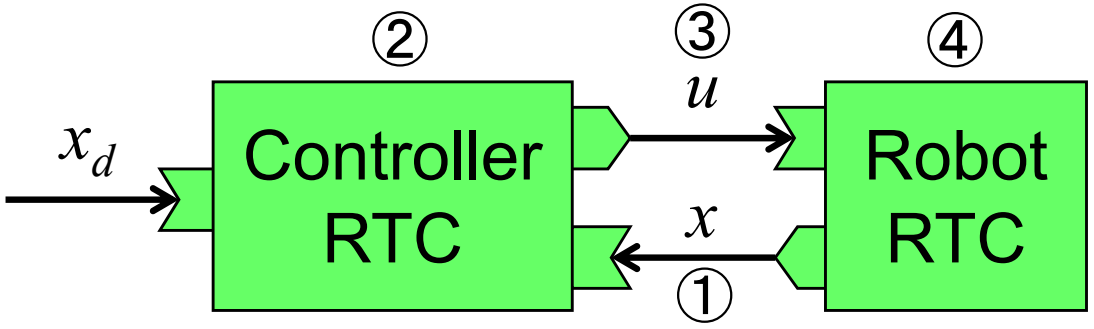
- 周期実行処理の順序
  - ① ロボットRTCが現在状態を送信する.
  - ② 制御器RTCが現在状態と目標状態を受信し，操作量を計算する.
  - ③ 制御器RTCが操作量を送信する.
  - ④ ロボットRTCが操作量を受信し，その操作量に従ってロボットを制御する.

この順序通り
指定周期で
実時間
実行する.



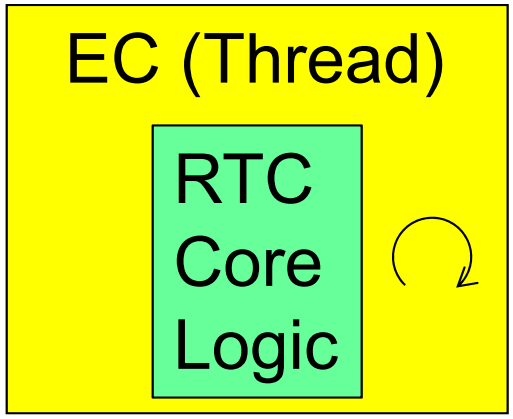
実時間制御を実現するための要件

1. 各RTCの制御処理が実時間で実行される.
2. 系全体の制御処理が規定の順序通りに実行される(RTCの同期実行).
3. データ送受信処理が実時間で実行される.



RTCの実行方法

- RTCのコアロジックは実行コンテキスト(EC)で駆動される.
- ECの実体はスレッド.
- 周期ECの場合，周期的にコアロジックを駆動.

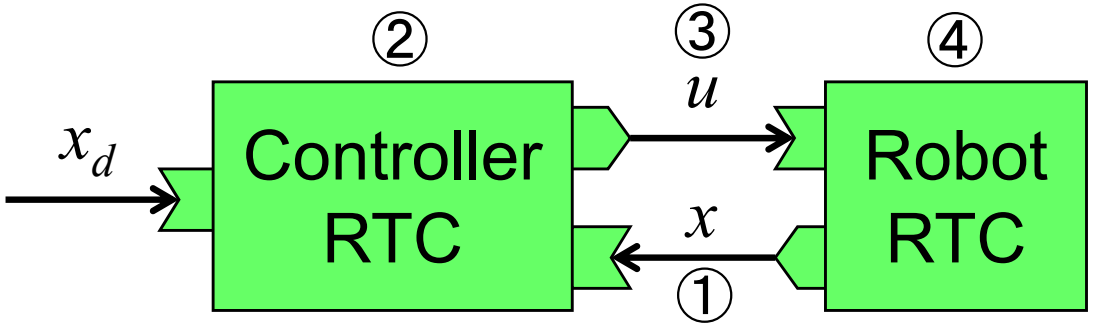


RTCの実行の実時間化

- 方法
  - 実時間OSの導入
    - ART-Linux
    - Linux with RT-Preempt, etc.
  - スレッド(EC)の優先度を上げる.
- 実現例
  - OpenRTM-aist-1.1.0(C++)で提供されているEC
    - ArtExecutionContext (for ART-Linux)
    - RTPreemptEC (for RT-Preempt Linux)

実時間制御を実現するための要件

1. 各RTCの制御処理が実時間で実行される.
2. 系全体の制御処理が規定の順序通りに実行される(RTCの同期実行).
3. データ送受信処理が実時間で実行される.



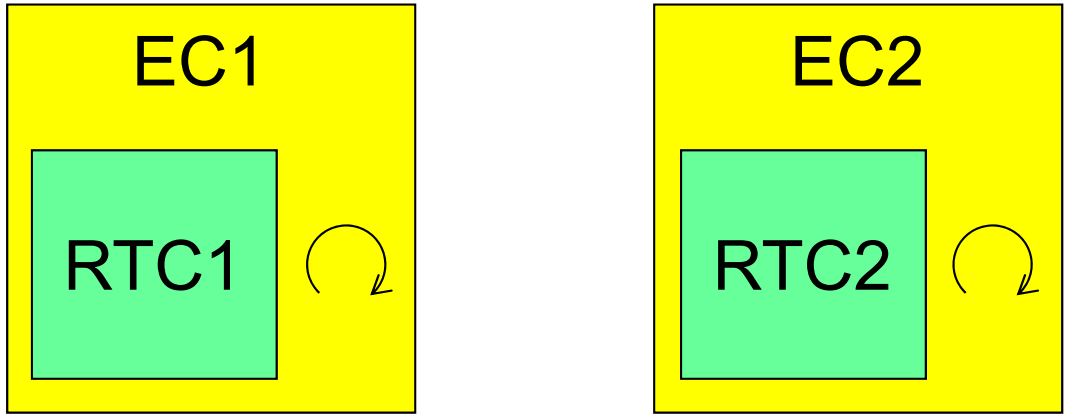
複数RTCの実行

- 通常，各RTCはそれぞれ異なるECで駆動される.

  ＝ 各RTCの実行は並列で非同期.

  ⇒ RTCの実行順序を制御できない.

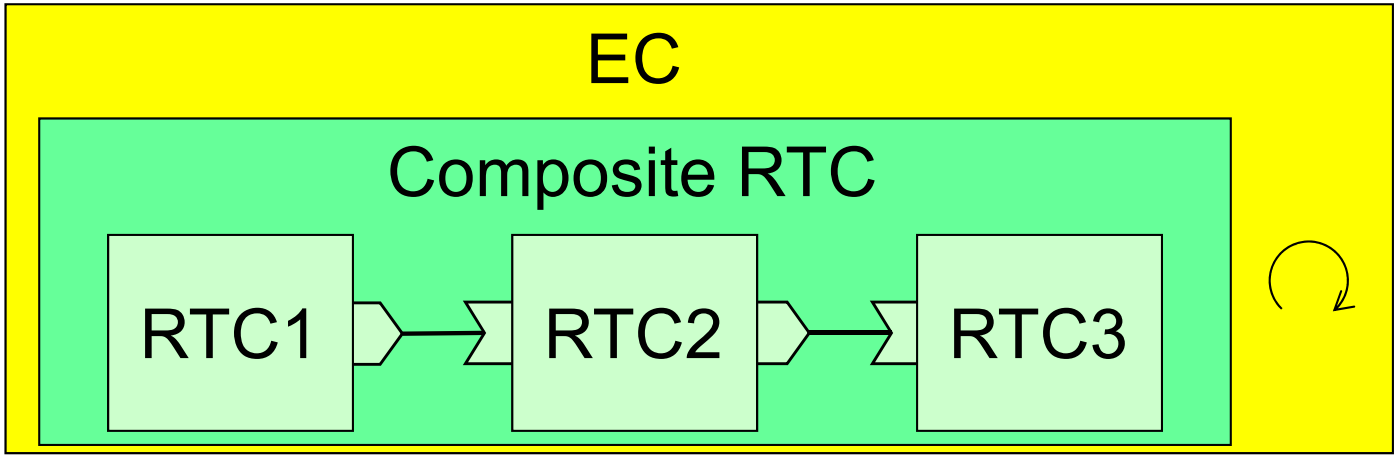
RTCを同期実行する方法が必要.



各ECの動作は互いに依存しない

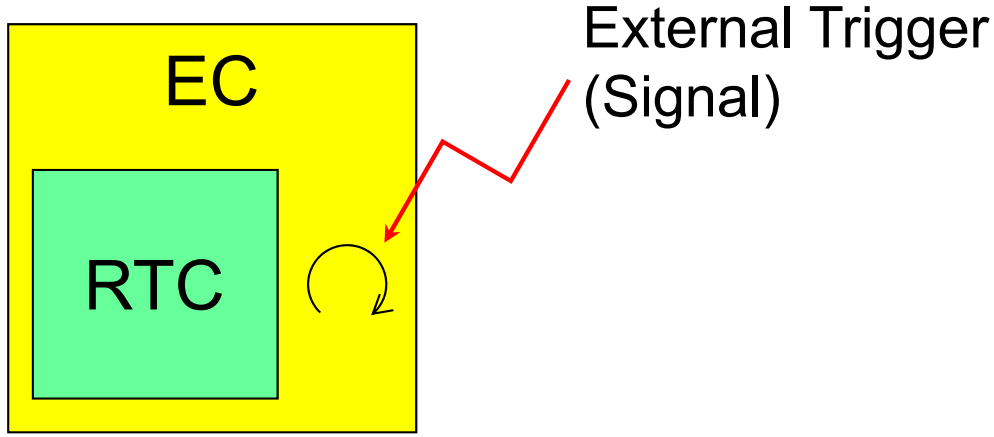
同期の実現方法(1)

- 複合コンポーネントを使う.
  - 内部のRTCは直列に実行される.
    ＝ 同期実行が可能.
  - 内部のRTC全てが同一プロセス上で動作していなければ，同期実行の実時間性は保証されない（コアロジックの駆動にCORBAを用いた場合）.



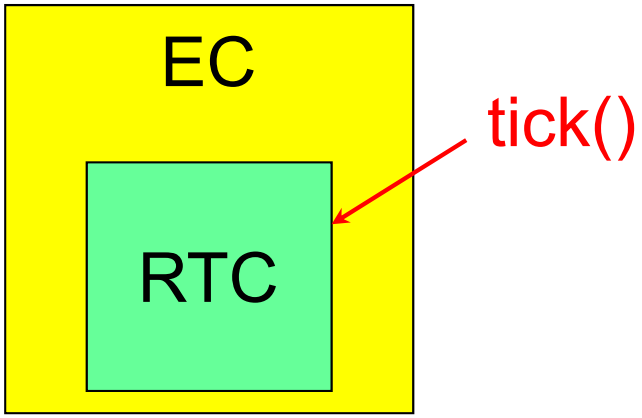
同期の実現方法(2)

- 外部トリガー実行コンテキストを使う.
  - OpenRTM-aistにおけるExtTrigExecutionContext.
  - RTCの実行は外部トリガーに同期.
  - 外部トリガーにシグナルを用いる場合，同期実行の実時間性は保証されない.



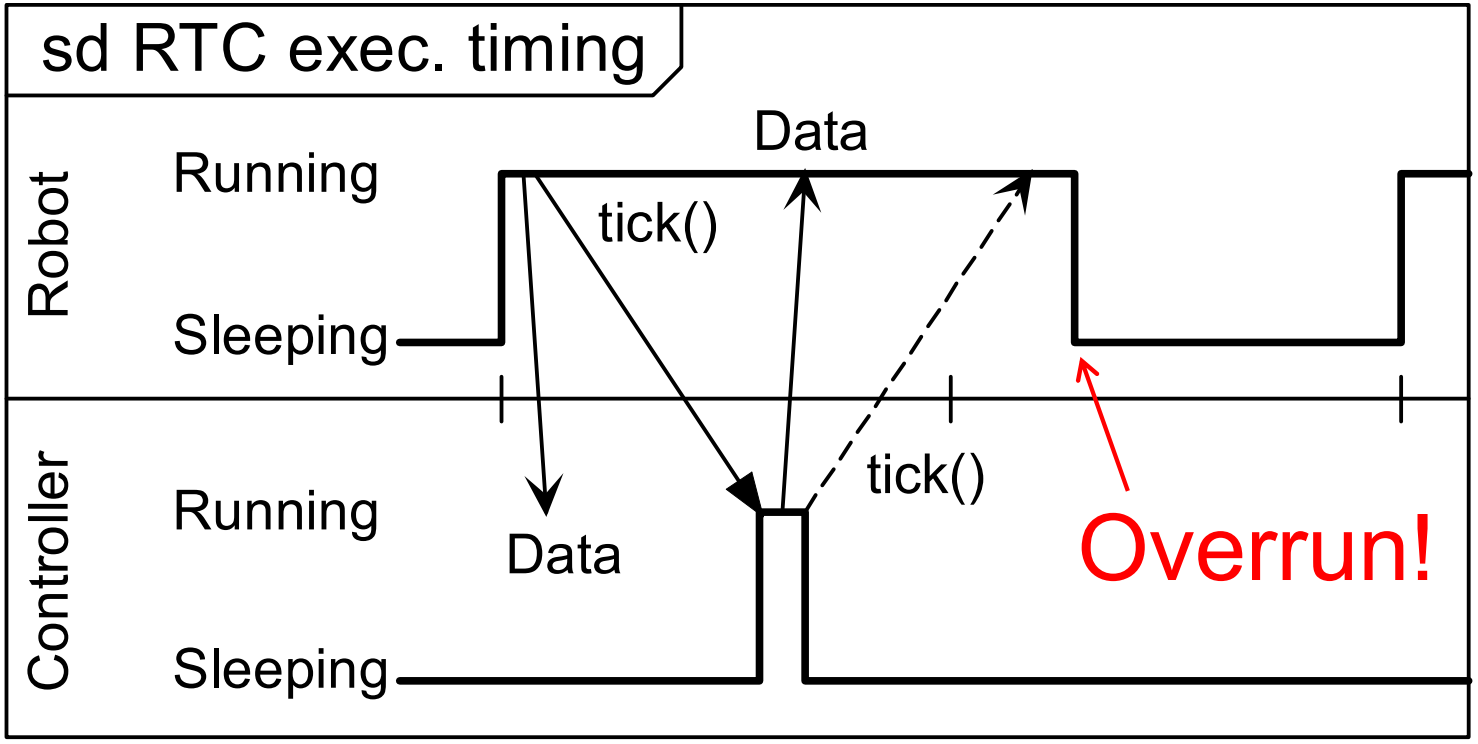
同期の実現方法(3)

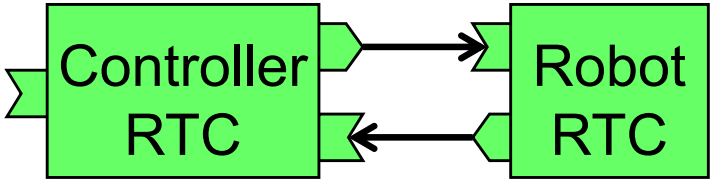
- 同期実行コンテキストを使う.
  - OpenRTM-aistにおけるSynchExtTriggerEC.
  - 同期実行が必要なRTCのコアロジックを直接駆動する.
  - コアロジックの駆動にCORBAを用いる場合，実時間性が保証されない.



非実時間同期ECの問題点

- タイミング図



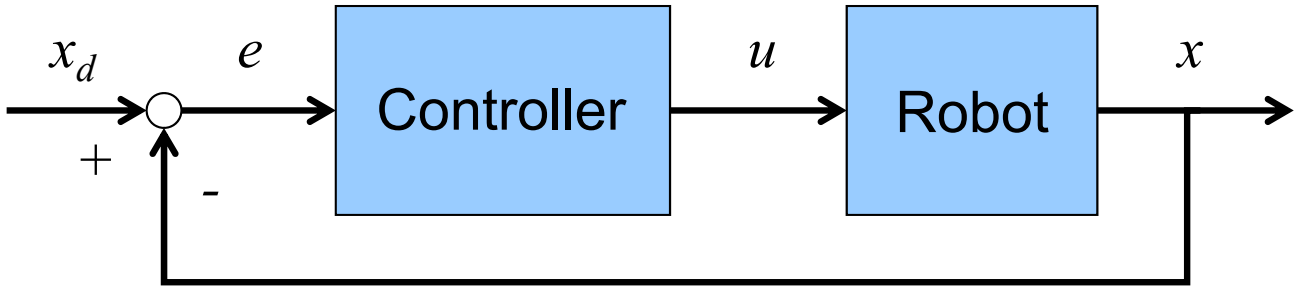


非実時間CORBA呼び出しのtick()
に長時間費やされるため,
指定周期内に処理を完了できない.

同期の実現方法(4)

- RTCの実行をデータに同期させる[1].
  - 制御系の構成要素はSISOシステム.
  - 各要素はデータ入力があった時だけ処理を実行し，処理結果のデータを出力するだけでよい.

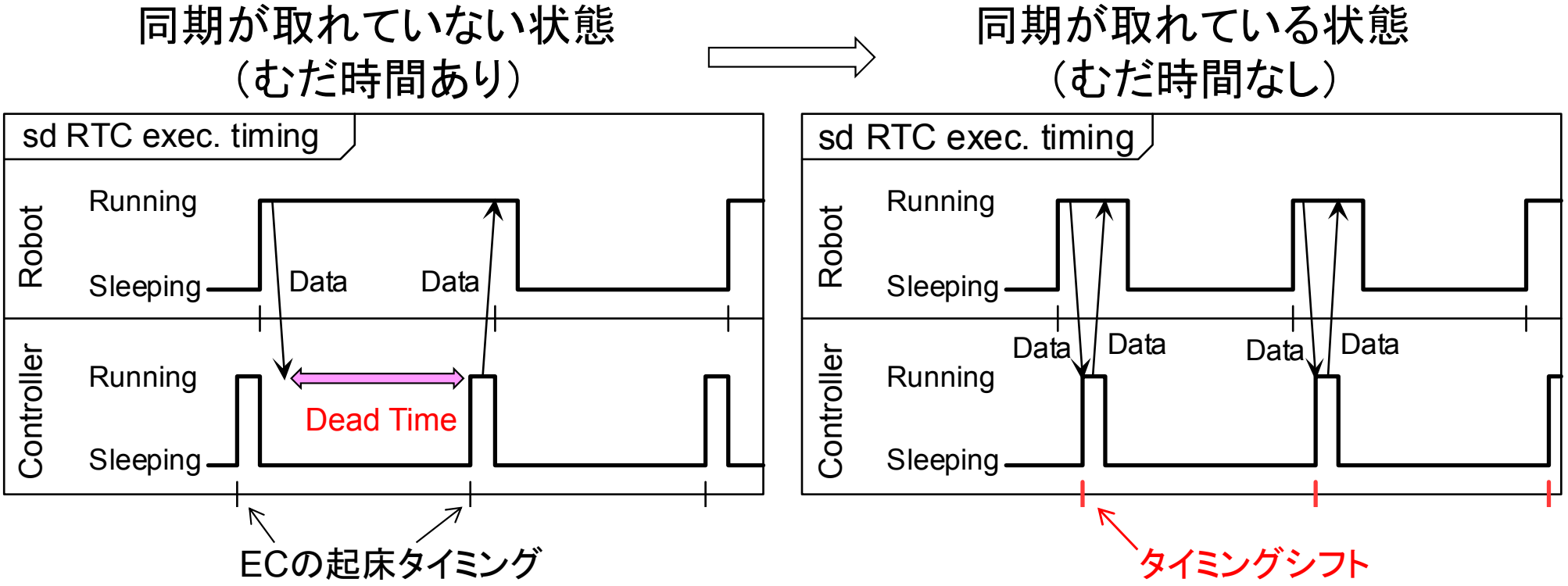
入力データに同期してRTCを駆動するECが必要



[1] 清水，石綿，尹，加賀美，”ART-LinuxにおけるRTコンポーネント間の同期方法の検討”，Robomec2013.

データ同期実行コンテキストの設計

- 基本アイディア
  - ECの周期タイマが起床してからデータが到着するまでの時間分だけタイマ起床時間をずらす.



データ同期実行コンテキストの実装例

- 実時間OSとしてART-Linuxを用いる.
- データ到着待ちには，art_wait_phase()を用いる（スリープしながらデータ到着を待つ）.
- ECのタイミングずれ時間はPCのクロックで計測する.
- ECのタイミングずれ補正量は，データ到着時間のばらつきを考慮して決定する.

$$t_{adj} = \begin{cases} -\Delta t & (\hat{\mu} = 0 \text{ and } \hat{\sigma} = 0) \\ \hat{\mu} - 2\hat{\sigma} & (\hat{\mu} > 2\hat{\sigma}) \\ 0 & (\hat{\mu} \le 2\hat{\sigma}) \end{cases}$$

- タイミング補正には，art_shift()を用いる.

データ同期実行コンテキストの実装例

- 実行コンテキスト本体

  RTC::PeriodicExecutionContextの派生クラスにする.

  ArtDataSyncEC → PeriodicExecutionContext

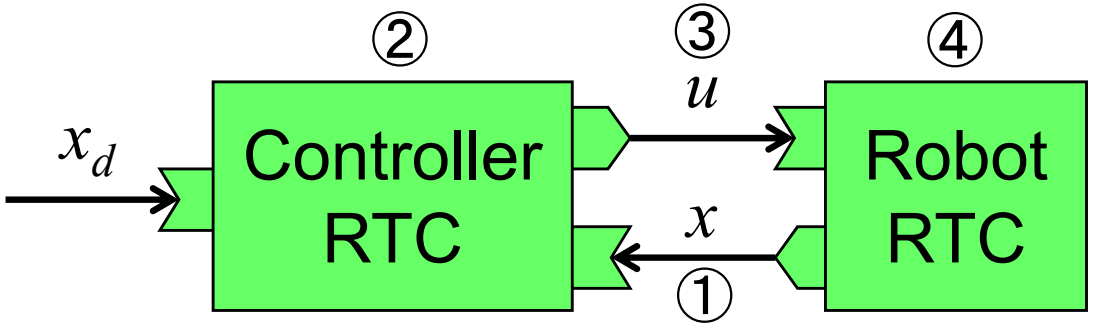
  ⇒ OpenRTM-aistにダイナミックロードできる.

- 同期データの指定

  データ入力ポートを登録するメソッドをECに追加.

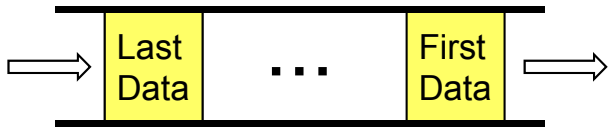
実時間制御を実現するための要件

1. 各RTCの制御処理が実時間で実行される.
2. 系全体の制御処理が規定の順序通りに実行される(RTCの同期実行).
3. データ送受信処理が実時間で実行される.

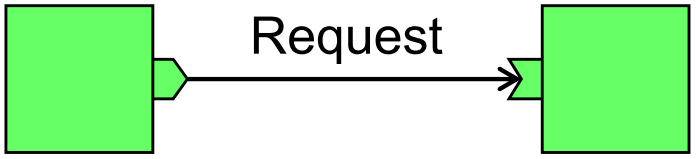


OpenRTM-aistにおけるデータ送受信方法

- OutPortからInPortへの一方向データフロー
- バッファ
  - データの生成と送受信のタイミングは非同期.
  - 一時的なデータ保管が必要.
- データフロー型
  - Push型
  - Pull型



FIFO Buffer

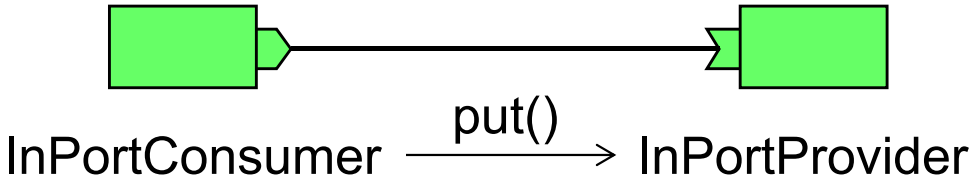


Push-type

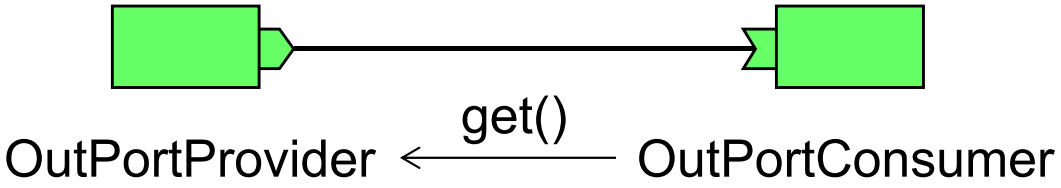
Pull-type

OpenRTM-aistにおけるデータ送受信方法

- データの送受信： サーバ・クライアント方式.
- プロバイダ（サーバ）とコンシューマ（クライアント）
  - Push型： InPortProvider + InPortConsumer



  - Pull型： OutPortProvider + OutPortConsumer



OpenRTM-aistにおけるデータ送受信方法

- OpenRTM-aist標準のプロバイダ・コンシューマ
  - Push型
    - InPortCorbaCdrProvider
    - InPortCorbaCdrConsumer
  - Pull型
    - OutPortCorbaCdrProvider
    - OutPortCorbaCdrConsumer
- データ送受信の実時間性
  - CORBAを用いているため実時間性が保証されない.

⇒ CORBAを用いないデータ送受信方法が必要

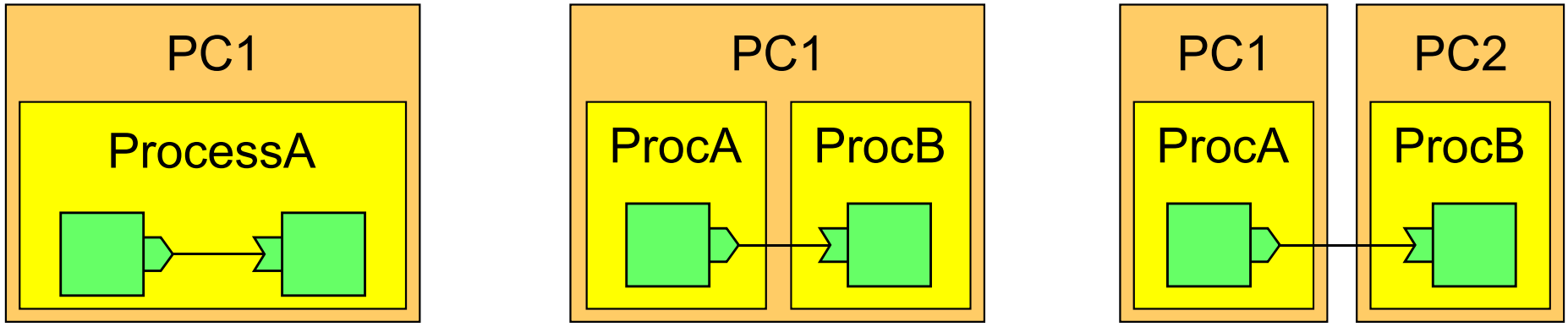
実時間データ送受信の実現方法

- RTCシステムの構成によって異なる.

  構成1　全RTCが同一PC上の同一プロセスで動作.

  構成2　全RTCが同一PC上で動作するが,
  各RTCは異なるプロセスで動作.

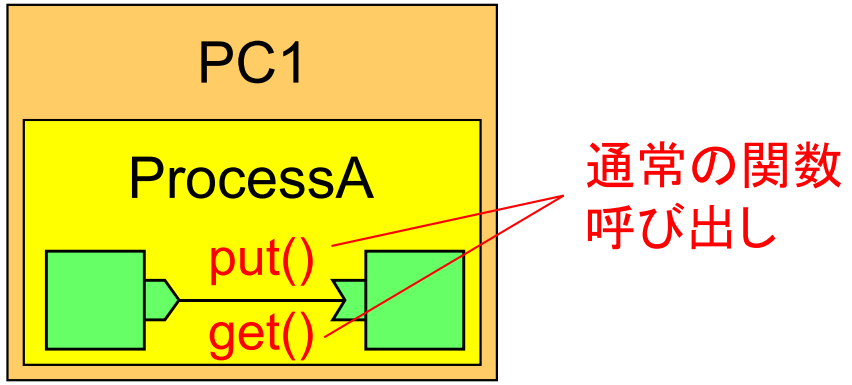
  構成3　各RTCが異なるPC上で動作.



同一PC，同一プロセス　　同一PC，複数プロセス　　複数PC，複数プロセス

実時間データ送受信の実現方法

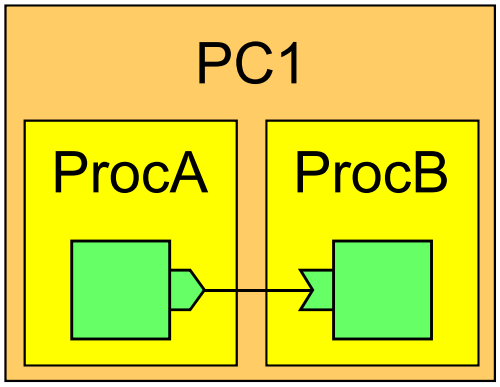
- 構成1の場合
  - CORBA実装にomniORBを用いれば，標準の方法でも実時間データ送受信を実現可能.
    - 同一プロセス内のCORBA呼び出しは通常の関数呼び出しに変換されるため.



実時間データ送受信の実現方法

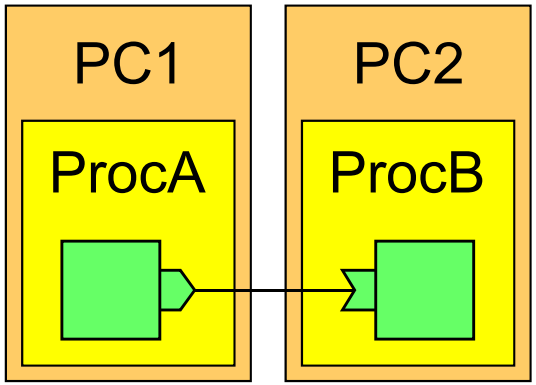
- 構成2の場合
  - データ送受信にはプロセス間通信が必要.
    - ソケット通信（CORBAで利用されている）
    - メッセージキュー
    - 共有メモリ，など

実時間性を考慮すると，共有メモリ方式が最適



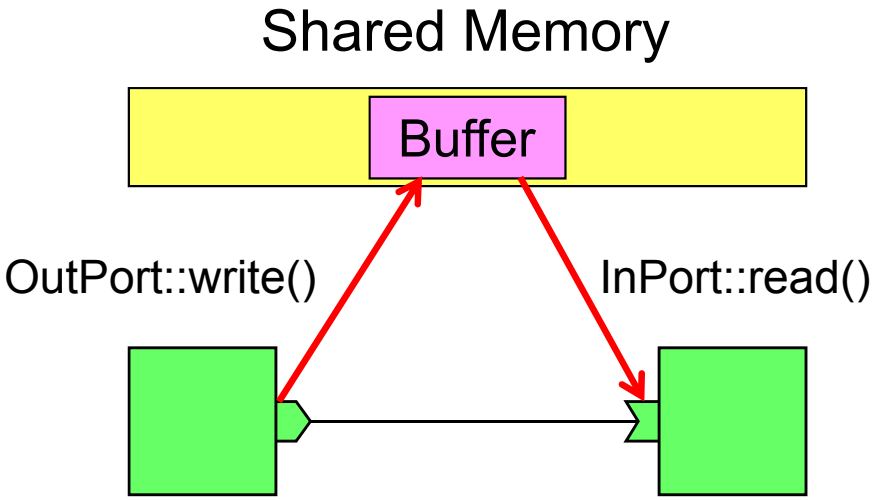
実時間データ送受信の実現方法

- 構成3の場合
  - データ送受信には通信ハードウェアが必要.
  - 実時間データ送受信を実現するためには，ハードウェアレベルの実時間性も考慮に入れる必要がある.
  - 実現例 ⇒ SI2013(2013年12月開催)で発表予定.



共有メモリを用いたデータ送受信方法

- バッファ
  - 共有メモリ上に作成する.
- データ送信
  - 共有メモリ上のバッファにデータを書き込む.
- データ受信
  - 共有メモリ上のバッファからデータを読み出す.



CORBAを用いないため実時間データ送受信が実現可能

共有メモリを用いたデータ送受信の実装例[2]

- 実時間OSとしてART-Linuxを利用する.
- バッファ
  - CdrArtShmBuffer → CdrBufferBase
- プロバイダ・コンシューマ（Push型）
  - InPortArtShmProvider → InPortProvider
  - InPortArtShmConsumer → InPortConsumer

各基底クラスの仕様に従ってモジュールを作成
⇒ ダイナミックロードでOpenRTM-aistに組み込める

[2] 清水，石綿，尹，加賀美，”AMP版ART-Linuxの共有メモリを用いたRTコンポーネント間のデータ通信の実現”，SI2012.

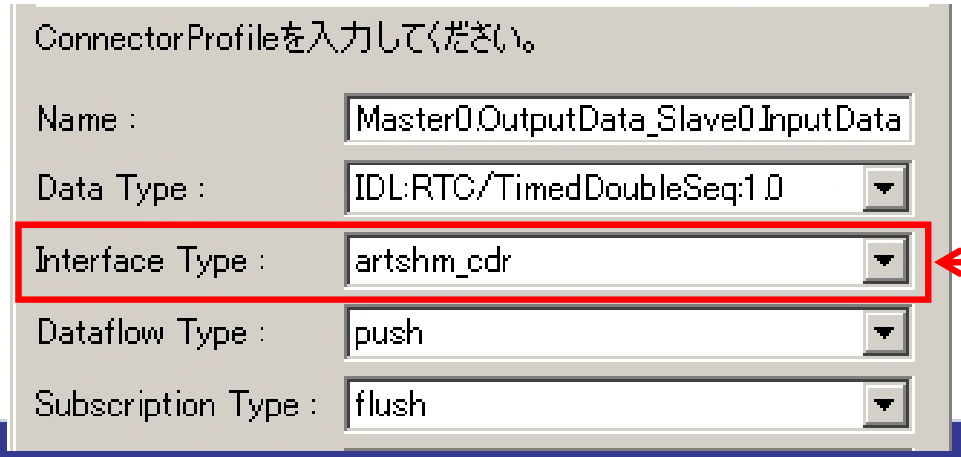
開発したモジュールの使用法

1. EC，バッファ，プロバイダ，コンシューマのロードとECの指定.
   - rtc.confに記述する.

     exec_cxt.periodic.type: ArtDataSyncEC

     manager.modules.preload: ArtDataSyncEC.so, CdrArtShmBuffer.so, InPortArtShmProvider.so, InPortArtShmConsumer.so

2. ポート接続時にインタフェースを指定する.
   - RTSystemEditorの場合

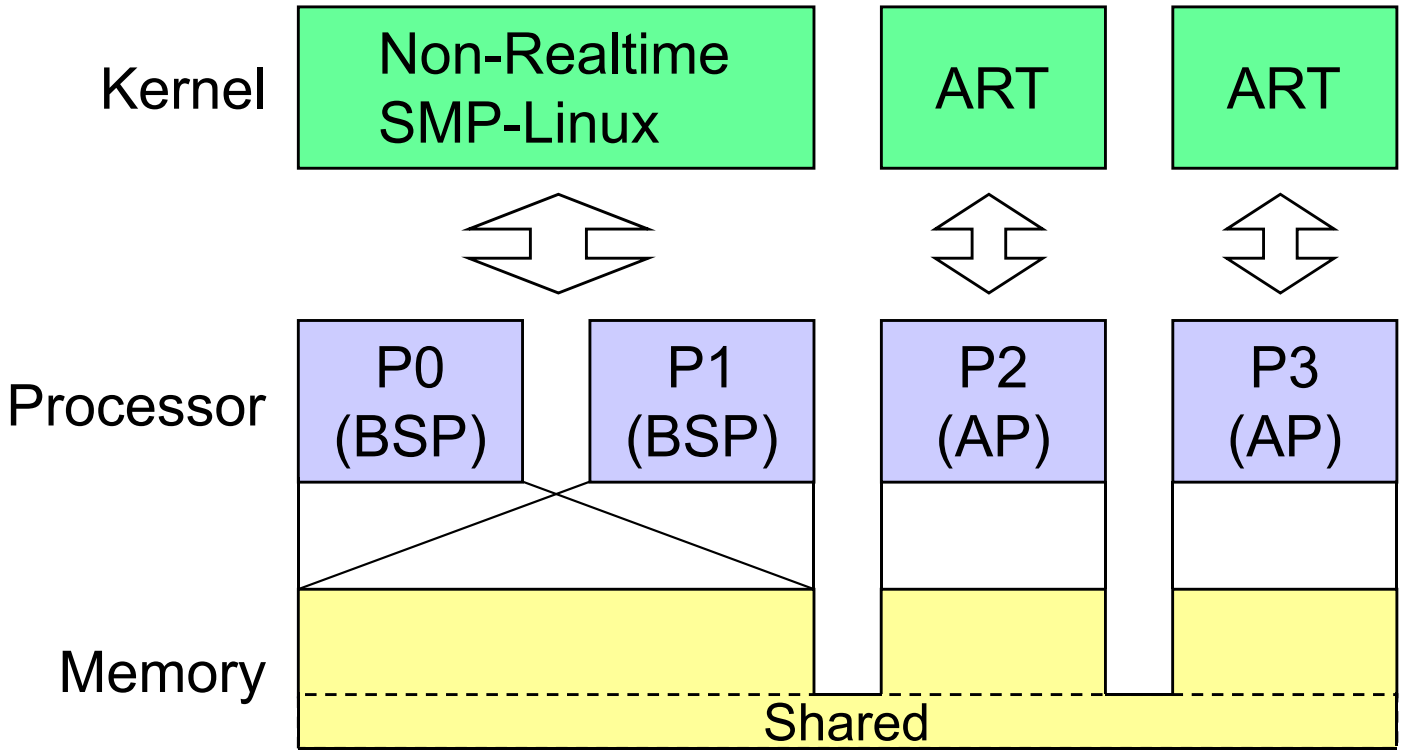


artshm_cdrを選択する.

ART-Linuxを用いた実時間性能試験

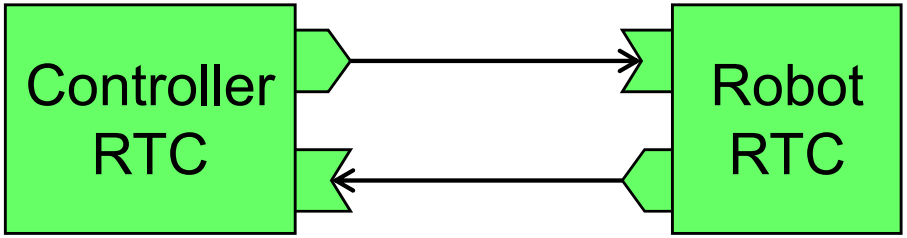
- AMPに対応したART-Linuxの概要

http://www.dh.aist.go.jp/jp/research/assist/ART-Linux/



ART-Linuxを用いた実時間性能試験

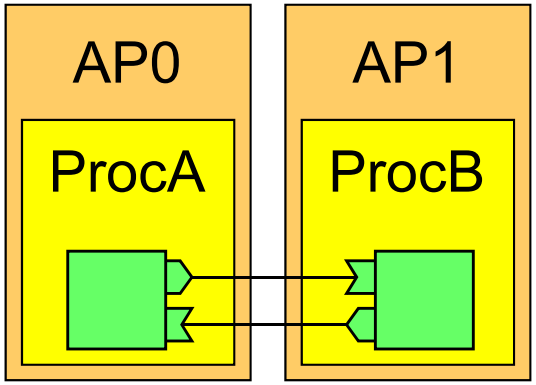
- 使用したシステム



  - システムの実行順序
    1. ロボットRTCがデータを送信する.
    2. 制御器RTCがデータを受信する.
    3. 制御器RTCがデータを送信する.
    4. ロボットRTCがデータを受信する.

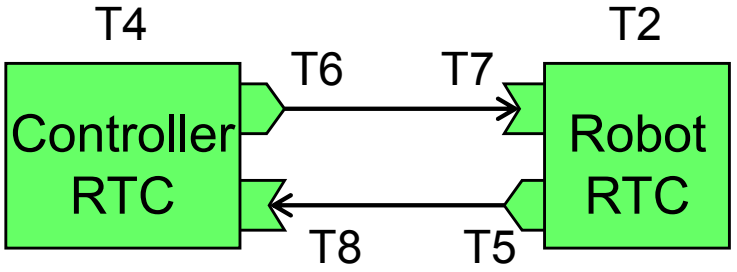
ART-Linuxを用いた実時間性能試験

- システム構成



- 実験条件
  - 制御周期：1ms
  - 実行コンテキスト
    - ロボットRTC: ArtExecutionContext
    - 制御器RTC: ArtDataSyncEC
  - データ送受信方法：共有メモリ方式

ART-Linuxを用いた実時間性能試験



- 測定した時間

T1: 制御系全体の実行周期

T2: ロボットRTCの周期処理の実行時間

T3: 制御器RTCからのデータ到着待ち時間

T4: 制御器RTCの周期処理の実行時間

T5: ロボットRTCから制御器RTCへのデータ送信時間

T6: 制御器RTCからロボットRTCへのデータ送信時間

T7: ロボットRTCにおけるデータ受信時間

T8: 制御器RTCにおけるデータ受信時間

ART-Linuxを用いた実時間性能試験

- 結果

（単位：μs）

	T1	T2	T3	T4	T5	T6	T7	T8
平均	1005	100	93	4	1	1	2	2
標準偏差	11	7	8	2	1	1	1	2
最大値	1029	112	106	16	6	4	12	12

- 系全体の実行周期T1は制御周期とほぼ同じ.
- 系全体の実行処理時間T2は最大で112μ秒

  ＝ すべての処理が規定時間内に完了.

提案方法で実時間制御を実現できる.

ART-Linuxを用いた実時間性能試験

- 比較実験の条件
  - 制御器RTCのECをSynchExtTriggerECに変更.
- 比較実験結果

（単位：μs）

	T1	T2	T3	T4	T5	T6	T7	T8
平均	2011	1213	1205	6	2	1	2	2
標準偏差	8	57	56	0	0	0	0	0
最大値	2224	1616	1610	21	5	10	6	11

  - 系全体の実行周期T1は制御周期を大きく超えている.
  - 系全体の実行処理時間T2は最大で1616μ秒

＝ 規定時間内に処理を完了できていない.

内容

- RTミドルウェアを用いた実時間ロボット制御系の構築
  - 実時間制御の実現方法.
  - ART-Linuxを用いた実現例と実時間性能.
- ソフトウェア教育
  - 研究室配属学生(学部4年生)への導入教育の実施内容と結果.

学部生のプログラミングスキル

- 静岡大学工学部機械工学科のカリキュラム
  - プログラミング Ⅰ（学部2年前期）
    - データ型，for文，while文などの基礎文法の学習
  - プログラミング Ⅱ（学部2年後期）
    - 関数，ポインタ，ファイル入出力などの学習とプログラミング演習
  - 数値解析（学部3年前期）
    - 数値計算アルゴリズムの講義とレポートによる演習

⇩

プログラミング（C言語）は2年生の時にしか習わない．

学部生のプログラミングスキル

- 現状
  - 研究室配属（4年生）段階で，プログラミングについてはほとんど忘れている（何をやったかすら記憶にない）．
  - プログラミングの講義では，プログラム作成のためのヒントが与えられており，それが無ければ白紙からプログラムを作成することは難しい．
  - プログラムの設計については講義や経験が無いためできない．
  - C++やJava等のオブジェクト指向言語を習っていないため，オブジェクト指向の概念がわからない．

現在実施中のロボット研究への導入教育

- ロボット工学の学習（2か月間）
  - 運動学，動力学，制御理論などの基礎知識を英語の本を読んで習得する.
- マニピュレータ制御プログラミング（3か月間）
  - 運動学，PID制御系，軌道生成のC++プログラミング.
  - 関節速度制御器のRTCの作成.

現状は…
運動学の計算や軌道生成の理論の理解に時間がかかり過ぎて，C++プログラミングまで満足に到達できない学生がいる.

C++プログラミングの教育

- これまでに試した事と結果
  - C++プログラミングを始める前に，C++の基礎文法とプログラム設計のキーポイントを講義形式で解説.

    ⇒ プログラミングを全くしたことがない段階では，説明を聞いても実際のコーディングまでイメージできない.
  - 独学でC++プログラミングを学ばせる.

    ⇒ C言語ライクなプログラミングから抜け出せない．オブジェクト指向やカプセル化が理解できていない.
  - 見本のプログラムを与えて，独学させる.

    ⇒ 見本のキーポイントを理解できず，意味がわからないまま写すだけになる（応用力が身につかない）.

RTMの教育

- 現在の教育方法
  - OpenRTMのWebページのチュートリアルで独学させ，分からない所は質問に答える形で教える．
- 学生を観察して分かったこと
  - C++でオブジェクトベースのプログラミングが出来なければ，RTC開発までたどり着けない．
  - 逆に，C++でオブジェクトベースの開発が出来るようになれば，コンポーネント指向のシステム設計やRTC開発も独学でできるようになる．

まとめ

- RTMを用いた実時間ロボット制御系の構築手法
  - RTCの実行の実時間化
  - 複数RTC間の実時間同期実行
  - データ送受信の実時間化
  - ART-Linuxを用いた実装例と実時間性能の検証
- ソフトウェア教育
  - 学生のプログラミングスキルの現状.
  - 研究室配属学生への導入教育の実施内容と結果.