µT-Kernel講習会 初級編
第1編:組込みリアルタイムOS入門（2013年度改訂版）

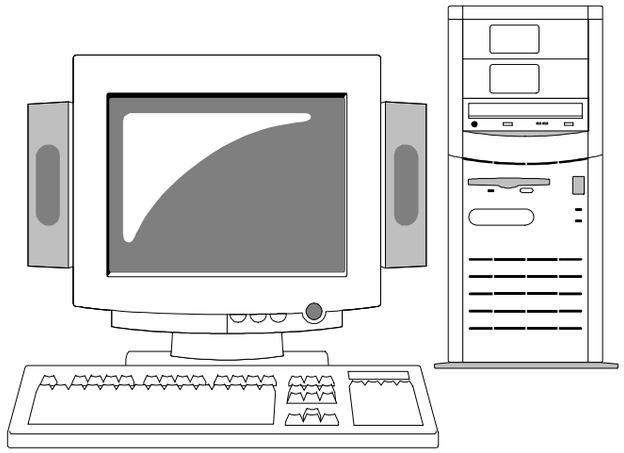
2013.12.18

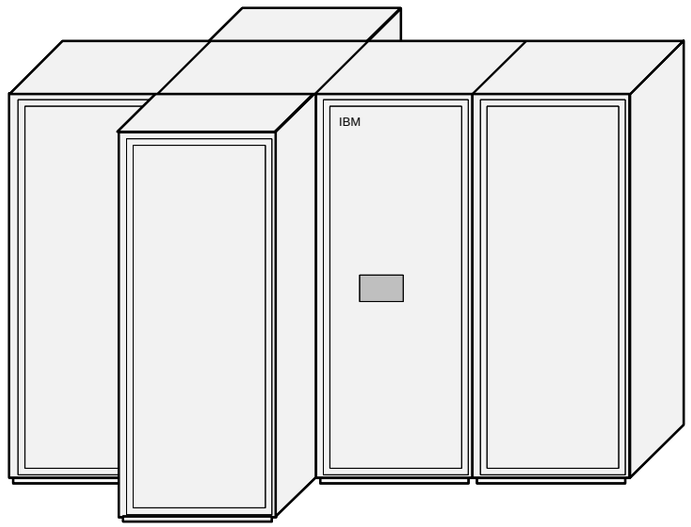
YRP Ubiquitous Networking Lab.
T-Engine Forum

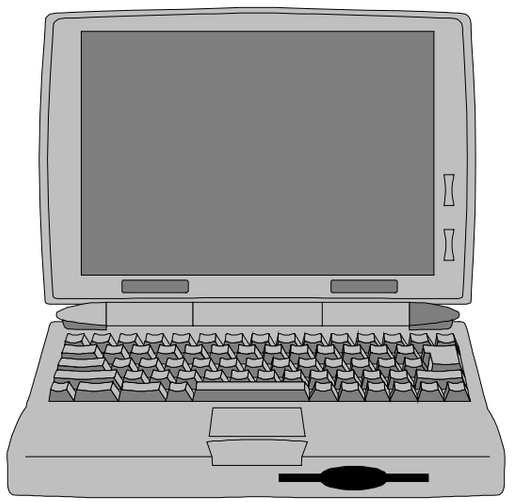
第1章
情報処理型コンピュータと組込みコンピュータ

情報処理型のコンピュータ（１）

- パーソナルコンピュータ
- サーバ
- スーパーコンピュータ



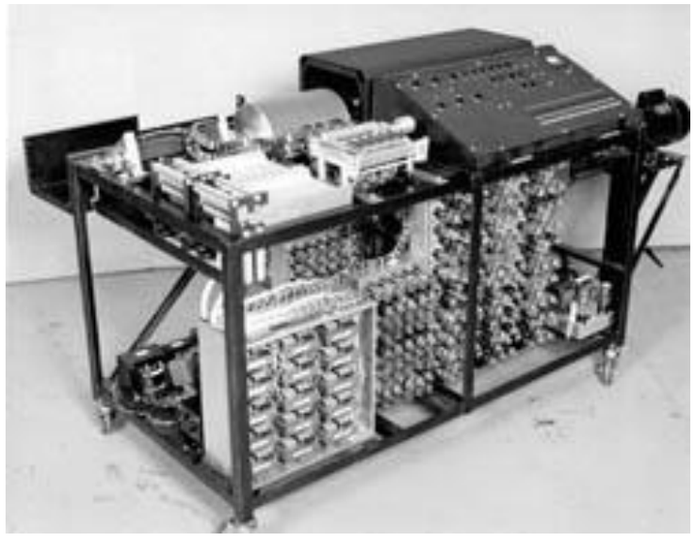




- まさに、コンピュータらしいコンピュータ
- コンピュータはこうしたものばかりではない！

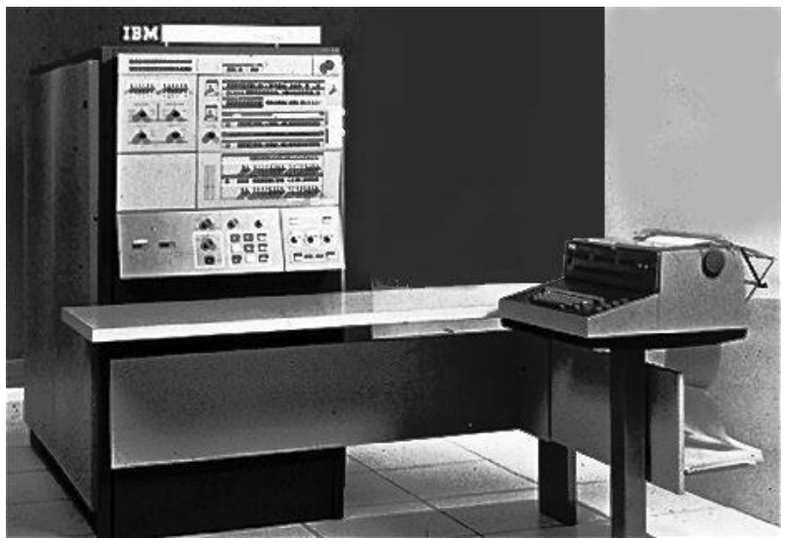
情報処理型のコンピュータ（２）

ENIAC



Atanasoff-Berry Computer

“K”（京）スーパーコンピュータ



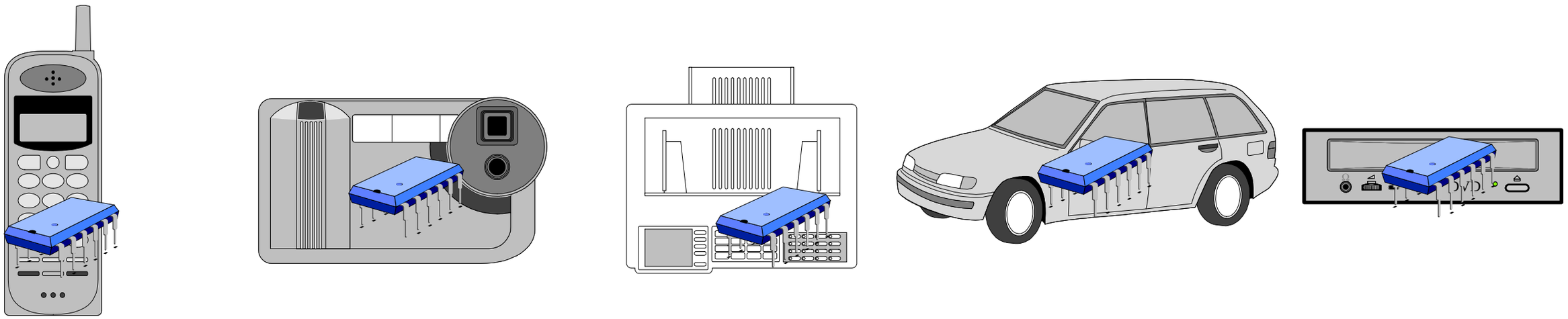
IBM 360

Cray 1

Apple II

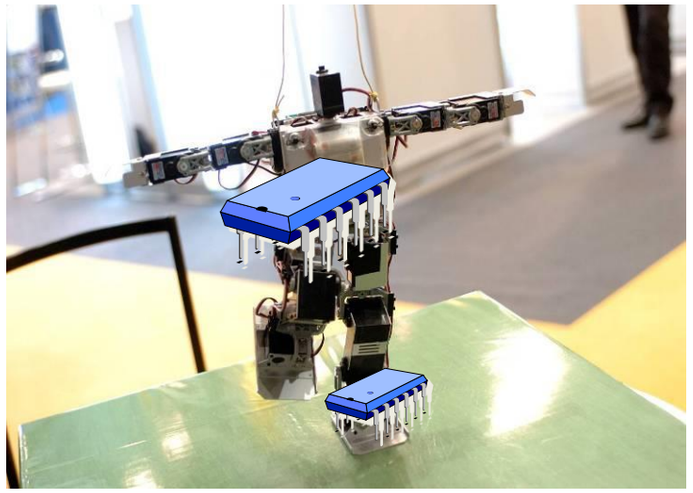
組込み型のコンピュータ（１）

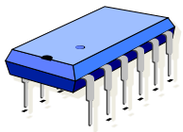
- 携帯電話
- ビデオカメラ
- デジカメ
- コピー機
- ファックス
- 自動車
- カーナビ
- MDプレイヤ
- DVDデッキ
- 自動販売機



これらにも全部、コンピュータが入っている
「組込みコンピュータ」（Embedded Computer）

組込み型のコンピュータ（2）





二つの種類のコンピュータ

- 情報処理型コンピュータ
  - いわゆる、「コンピュータ」らしいコンピュータ
  - 主な目的は、「情報」を扱うこと。
    - 情報＝数値、文字、絵、音声、動画、…
  - 人間に例えると、「頭脳」型コンピュータとも言われる。

- 組込み型コンピュータ
  - 最終形が、「コンピュータ」と呼ばれないものに組込まれているコンピュータ
  - 主な目的は、実世界の中で、「機器」を制御すること。
  - 人間に例えると、「反射神経」型コンピュータとも言われる。

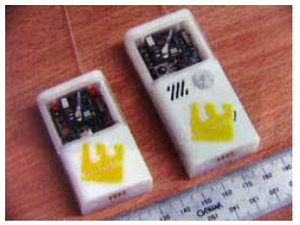
組込み型コンピュータこそメジャー

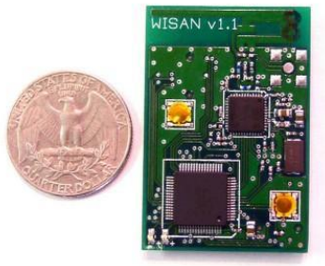
		2012 (Units)	
MCU Market	4～8bit	6,343 M	
	16bit	7,227 M	
	32bit	3,700 M	
	Sub Total	17,271 M	組込系
Computing Market	Server	9 M	
	PC	355 M	
	Mobile	653 M	
	Sub Total	1,017 M	非組込系

ネットワークに溶け込む組込み

背景：通信機能を持った超小型デバイスとクラウド

クラウドコンピューティング用
サーバーシステム





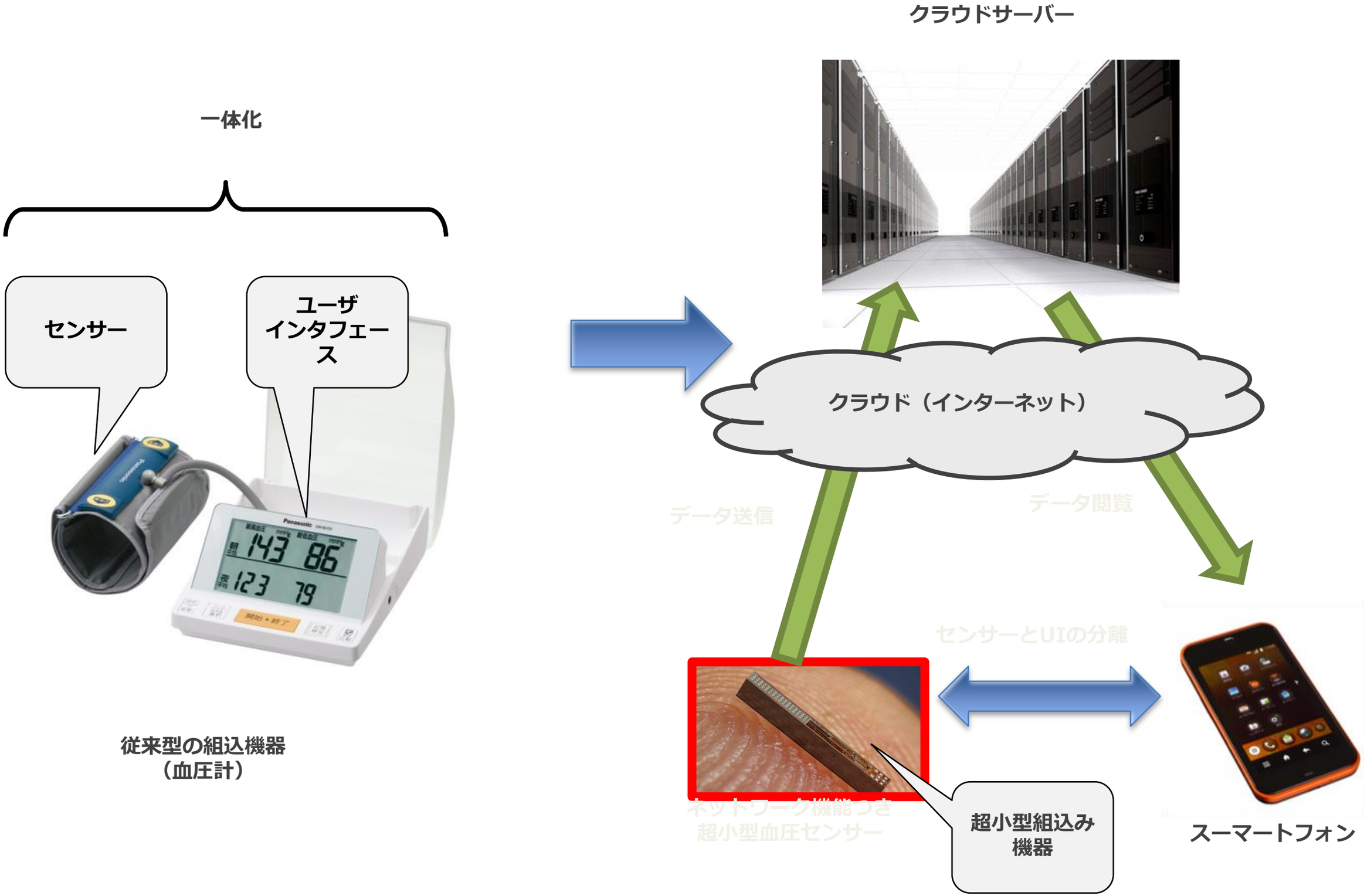
通信機能を備えた
超小型デバイス

（例）Google Cloud Print (with EPSON)

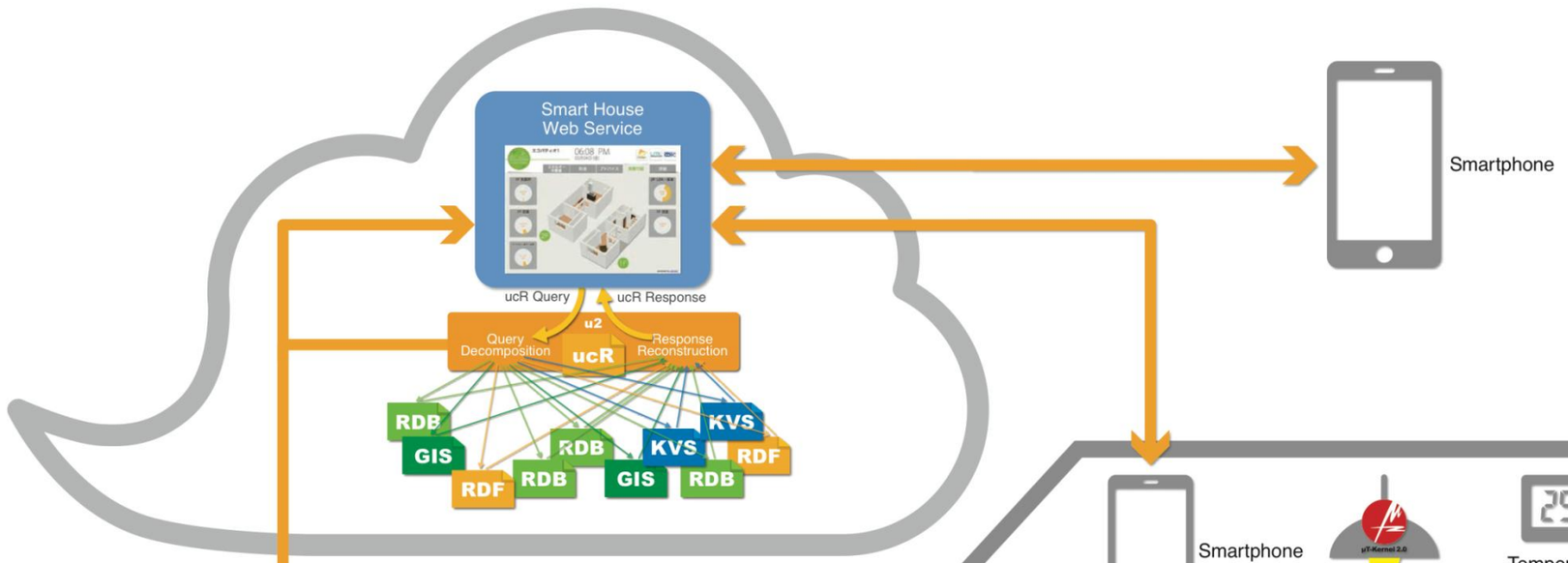
http://www.epson.jp/connect/gcp/

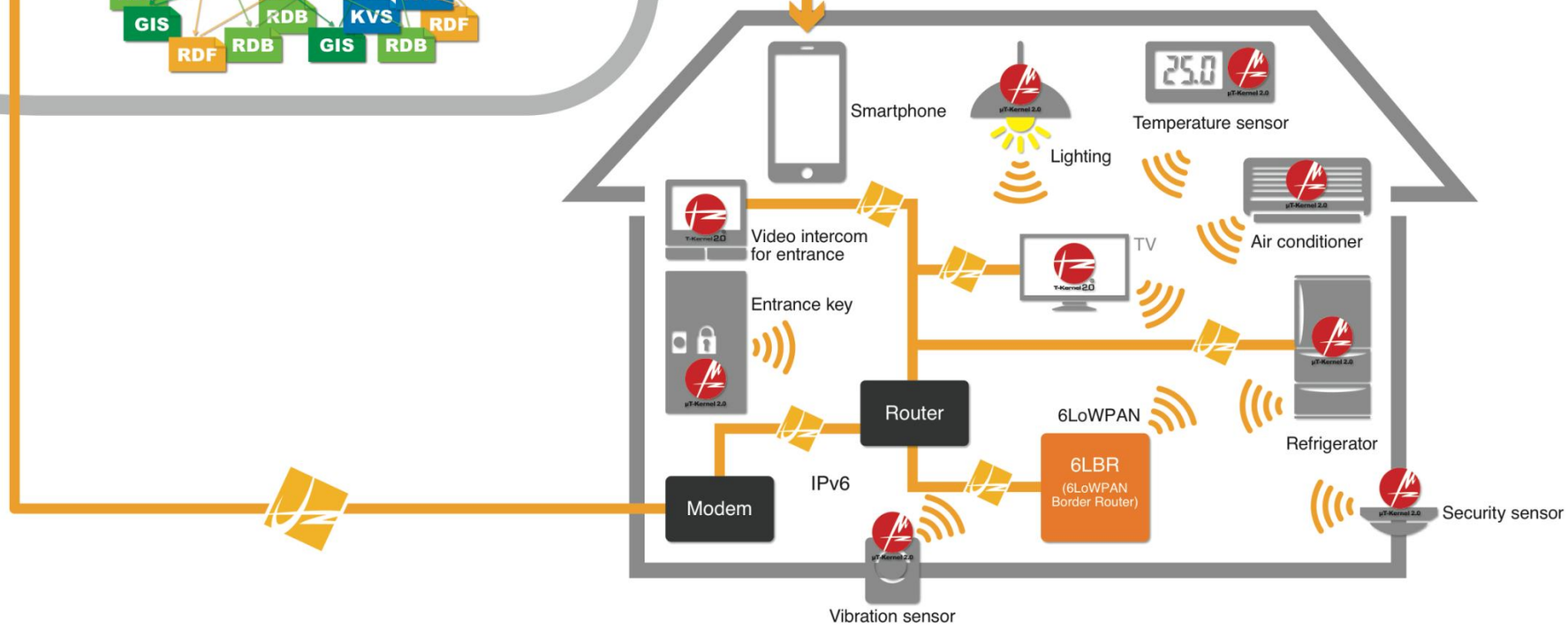
（例）クラウド対応体組成形（Gooからだログ）

M2M, IoT型組込みアーキテクチャ

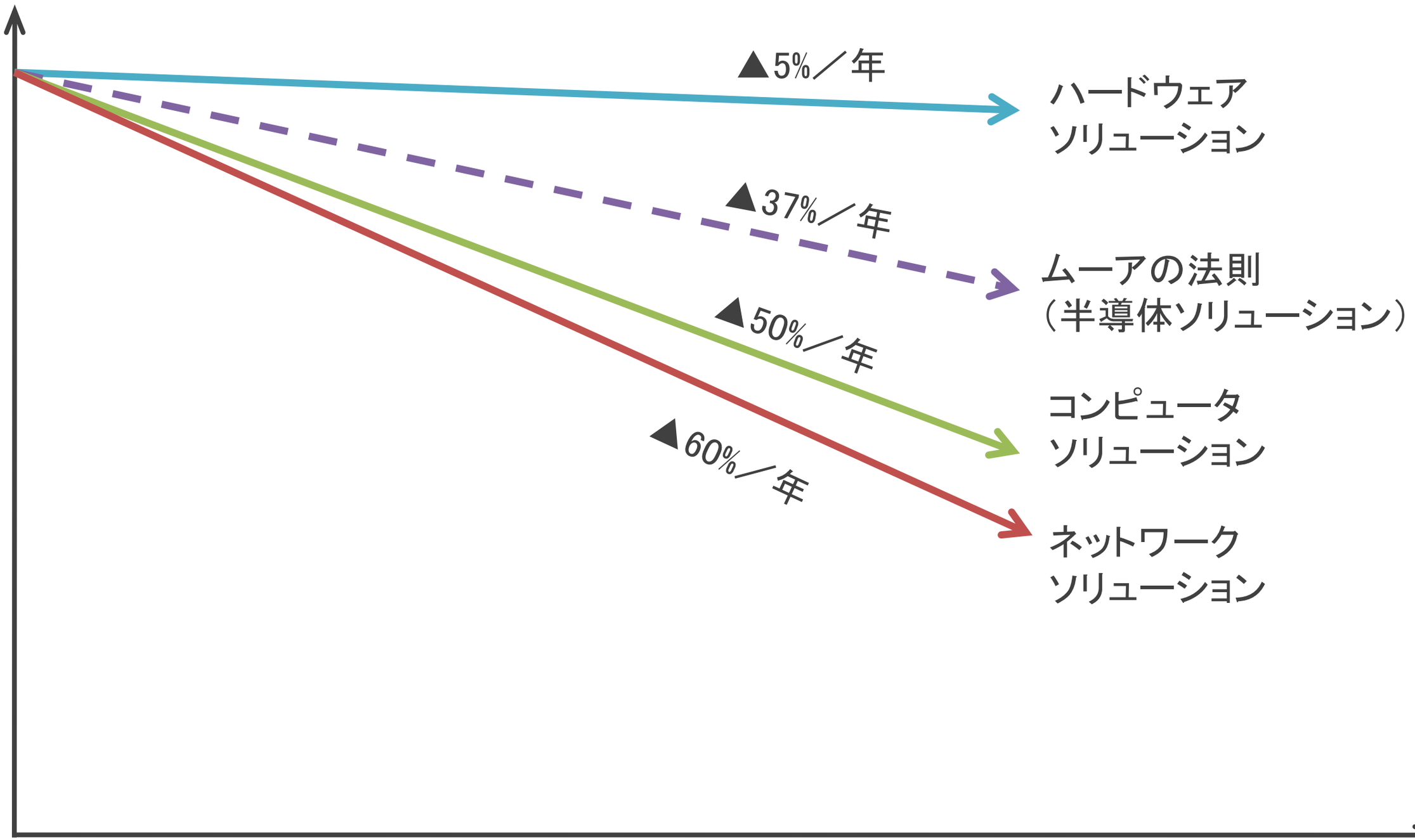


クラウドやスマホと一体化する家電





コストベネフィット



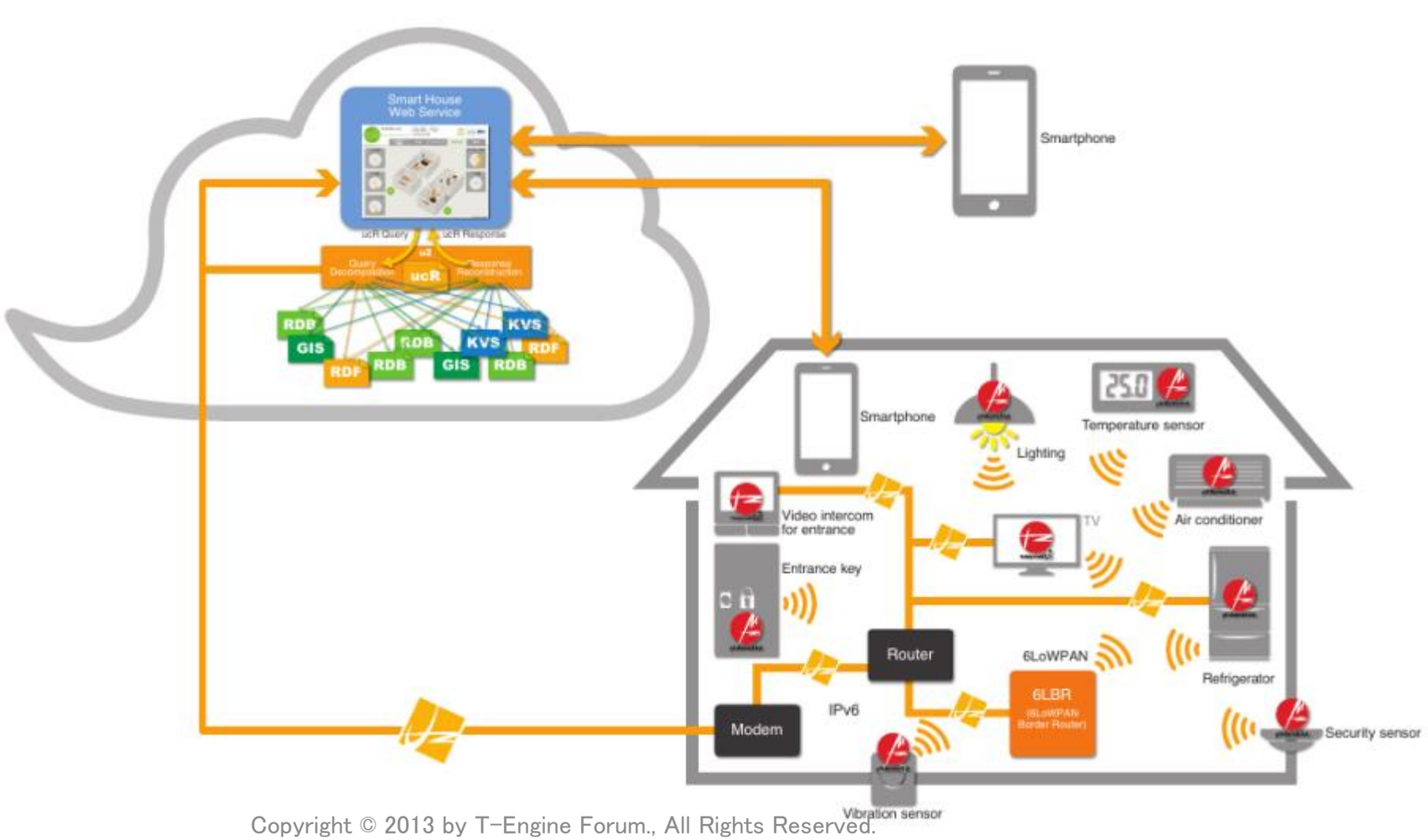
μT-Kernelとは？

μT-Kernelとは？

- T-Kernelの小規模MCU向けバージョン
  - 16ビットMCUや、ROM/RAMの限られた環境が対象
  - 小規模MCUでも利用したいという要求に応えるために設計されたのが「μT-Kernel」
- T-Kernelより弱く、μITRONより強い仕様
  - 開発効率向上のための“強い仕様”と、適応化・最適化の許容という相反する2つの要求のバランスをとった
    - T-Kernelと異なり、ソースコードの一本化は行わない
    - μITRONより大きな標準化範囲を持ち、リファレンスコードが存在

µT-Kernelの “2.0” 化

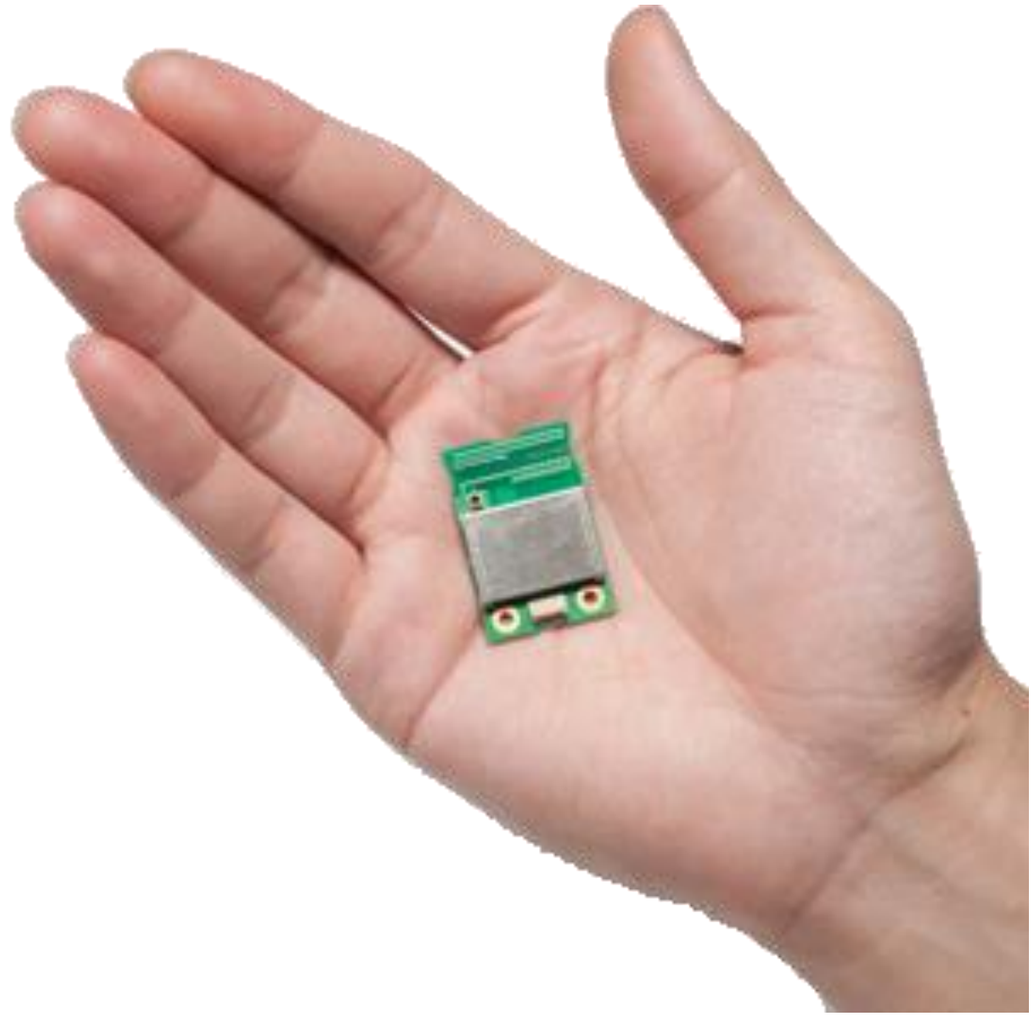
- T2全体コンセプト
  - 1984年に開始したTRONプロジェクトの目標である HFDS（超機能分散システム）を実現するための全体アーキテクチャの要素としてデザイン
  - ėネットワーク機能の強化と、それに基づく徹底した機能分散のためのリアルタイムOS



IoT (Internet of Things) のために

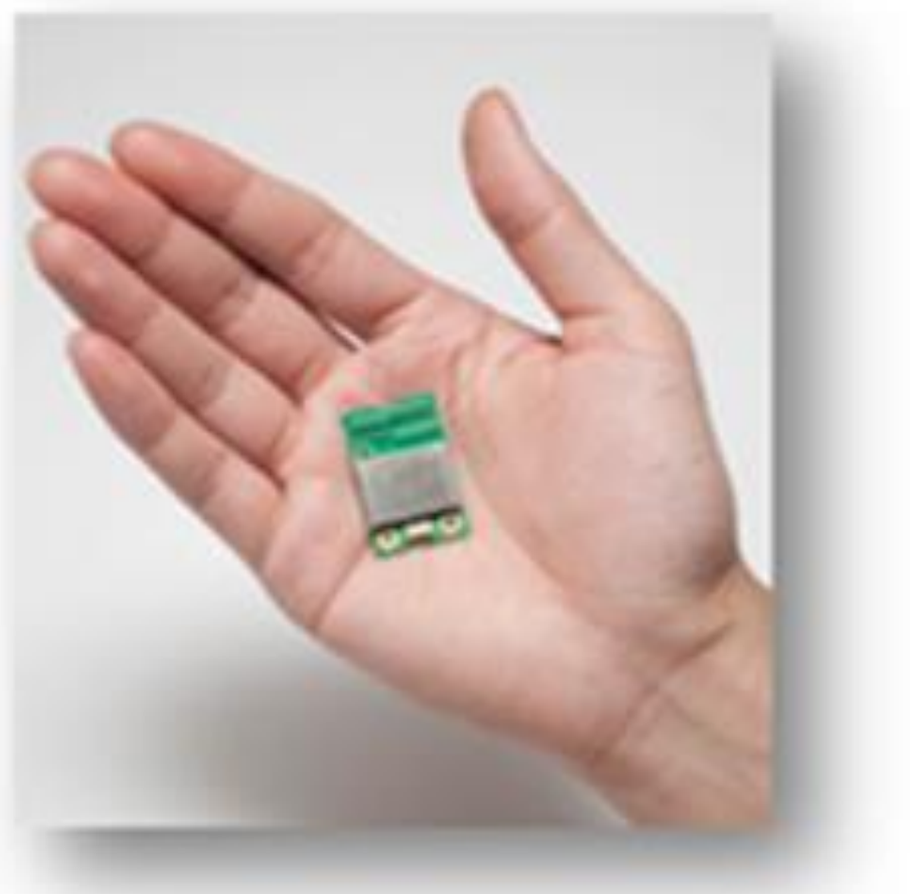
- μT-Kernel 2.0の位置づけ
  - M2M, IoTノード（小規模家電を含む）のためのリアルタイムOS

- そのための機能を凝縮
  - ネットワーク通信機能を含むIoTの為のミドルウェアが必要
    - 開発効率向上のためにはミドルウェア流通が可能なように
  - 省資源・省電力は必須
    - （例）バッテリだけで10年以上動作するセンサノード



µT-Kernel 2.0 (µT2)

- 小規模MCUのための最新リアルタイムOS
  - ”T2”コンセプトに基づく最新版
  - ROM: 8KB・RAM: 4KB 程度の小規模システムでも動作
- 特徴
  - サービスプロファイルの導入
    - MCUごとに最適化されたµT2間で、互換性を向上させる仕組み
  - 最適化・省資源化のさらなる追求
    - より少ないRAM/ROMでの実装を可能とする仕様の改訂



TO T1 T2

1984 2002 2011 2012 2013 2014～

T-Kernel Extension

- Multicore/ Multiprocessors
- 32-bit Processors
- 8/16-bit Processors

Process-based Programming

Task-based Programming

T-Kernel

- Multicore/ Multiprocessors
- 32-bit Processors
- 8/16-bit Single Chip

Development Environment

TKSE

Extension for SMP

Extension for AMP

T2EX

μT2EX

SMP T-Kernel

AMP T-Kernel

MP T-Kernel 2.0

ITRON2

ITRON1

μITRON

μITRON3.0

μITRON4.0

T-Kernel

T2

μT-Kernel

μT-Kernel 2.0

Eclipse

GNU/GCC

One Stop Service

Emulator

第２章
「組込み」システムとは？

組込みシステムの定義

- 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…

組込みシステムの要件（従来からの要件）

- リアルタイムシステム
  - 計算処理よりも、入出力処理、通信処置が中心
  - モノを制御するため、高い応答性能が要求される
- 性能、サイズのチューニング
  - （特にハードウェア）コストを極限まで下げる
  - 結果として厳しいリソース制約上でソフトウェア開発
  - コンパクトな実装
- 専用化されたシステム
  - 必要のない機能を削除することでチューニング可能
- 高い信頼性
  - 組込みシステムは、クリティカルな応用の場面も多い
  - ネットワークアップデートなどの仕組みがないものは、システムの改修に多大なコスト

http://panasonic.co.jp/

Panasonic Japan

商品一覧 | サポート 検索キーワードを入力

商品や企業に関する様々な情報は、パナソニック・ホームよりご覧ください。

パナソニック・ホームへ →

東日本大震災からの復興を心よりお祈り申し上げます。

震災による電力不足と停電対策のための家電製品のお取扱いについての情報をご紹介しています。
より安全にお使いいただくためにぜひご活用ください。

- 震災により被害を受けられた当社製品の点検・修理について
- 停電や地震の影響でよくあるお問い合わせや節電に関する情報
- 東日本大震災における対応や支援活動について

いま一度、心からのお願いです

ナショナルFF式石油暖房機を探しています

1985年(昭和60年)から1992年(平成4年)製のナショナルFF式石油温風機及び石油フラットラジアントヒーターには事故に至る危険性があります。当該対象製品を未処置のままご使用になりますと、一酸化炭素(無臭)を含む排気ガスが、室内に漏れ出し、死亡事故に至るおそれがあります。

対象製品のお引き取り(1台あたり5万円お支払いいたします)、もしくは無料で給気ホース部の点検修理をさせていただきます。不使用の対象製品もお引き取りさせていただいております。

対象製品の番号など詳しくはこちらをご覧ください

FF式石油温風機

石油フラットラジアントヒーター

ご連絡先:パナソニック株式会社　FF市場対策本部
(旧社名:松下電器産業株式会社)

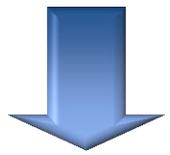
フリーダイヤル電話 0120−872−773(FF式石油暖房機受付専用)
受付時間 : 9時～17時(土曜日・日曜日・祝日を除く)
上記時間外につきましては、留守番電話にて受付させていただきます。

フリーダイヤルFAX 0120−870−779(FF式石油暖房機受付専用)

- FF式石油暖房機以外のお問い合わせ

組込みシステムの要件（新しい要件①）

- ユビキタス時代を迎え、組込み機器もネットワーク接続され、単体では動作しない。



- ネットワーク通信機能
  - サーバとの連携で、機能を実現
  - 他の製品やサービスとの連携
- セキュリティ
  - 暗号通信
  - 認証通信

組込みシステムの要件（新しい要件②）

- 省資源･省電力
  - 増えるモバイル型の組込み機器
  - バッテリーの持続時間は、製品競争力上重要
  - 世界的な省エネルギー意識の高まりから、省電力であることは重要

- 使いやすく、やさしい利用者インターフェース
  - 幼児からお年寄りまで。

組込み型コンピュータで特に重要な実時間性

- 身の回りの「組込みコンピュータ」の仕事は？
  - 給料計算や数値計算をするわけではない。

- 実世界の動きにあわせ、人間にサービス
  - 実世界の時間に合わせて動作する

- 「リアルタイム（実時間）システム」
  （Real-time System）

第3章
「リアルタイム」システム
Real-time System

リアルタイムシステムとは？

- 一般的には、次々に起こる実世界の事象（イベント）に合わせて素早く処理することが求められるようなシステム

- 入出力処理
- 機器類の制御
- 実時間の進行に追従した処理が重視

- きちんとした定義は意外と難しい（後のスライド…）

リアルタイムシステムを考える…

- 根本的には、「素早い」応答、実世界の実時間に追従

- では、計算処理性能が高いコンピュータであれば、リアルタイムシステムか？
  - 半分あたり、半分はずれ

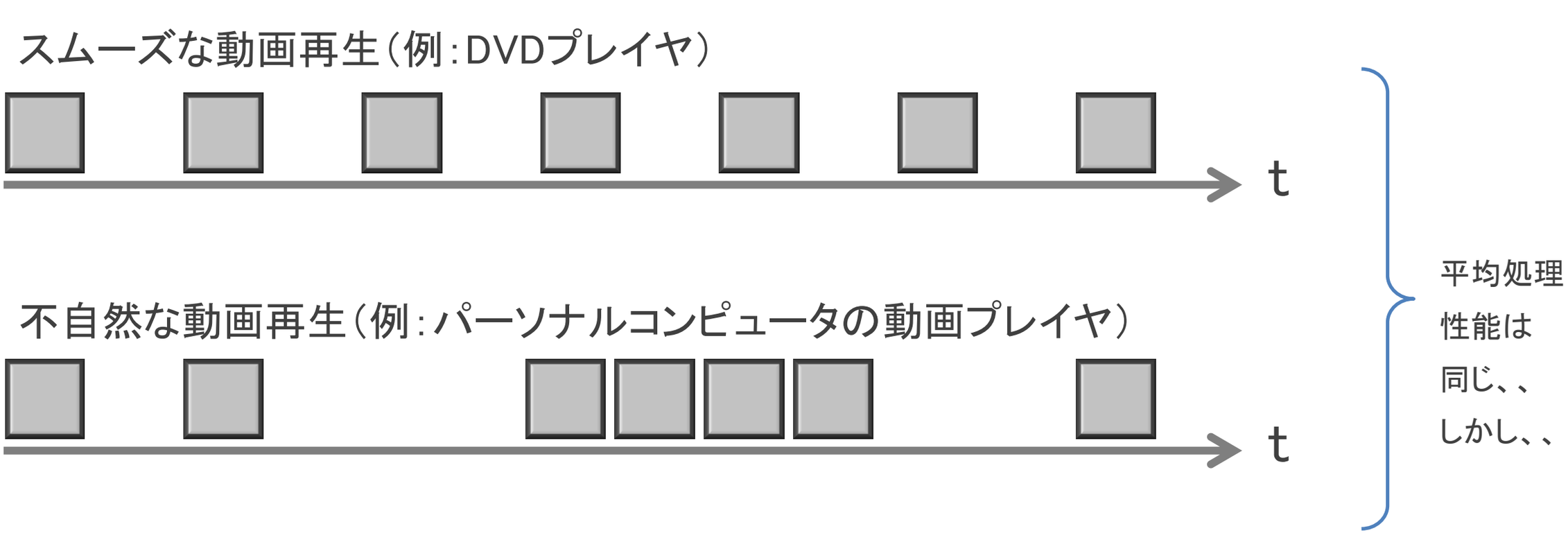
- 実世界の実時間に応答・追従のための「時間制約」を満たすために、コンピュータが十分「速い」ことは必要条件
  - 計算処理性能が高いコンピュータであるにも係らず、時間制約を満たせてないケースもある（つまり、十分条件ではない）
  - リアルタイム向きにできていないことが原因（次のスライド）
  - どこまで短い時間制約にまで対応できるかも、重要な指標（後述）

（例）時間制約を守る

ビデオの再生

- 現在のビデオ動画 ➔ 1秒間に30フレームを表示して再生

  ※ 1秒間に100枚の画像を表示する能力（平均処理性能）があっても、1/30秒に必ず1枚表示することができるとは限らない。



リアルタイムシステムの定義

- リアルタイムシステムは、利用できる計算機資源（resource）に限りがある中で、故障のような厳しい結果をもたらす応答時間制約を満たすことができるシステム
  - 「故障」＝システム仕様での要求を満たせないこと

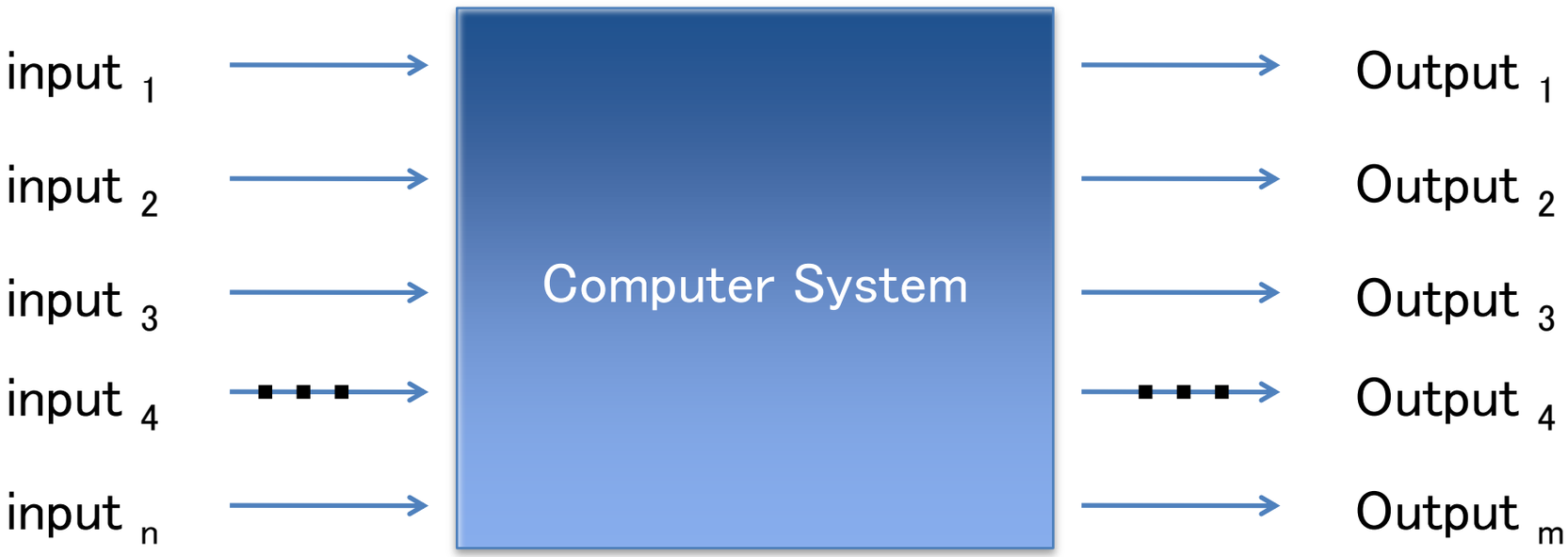
- リアルタイムシステムとは、その論理的正当性が、アウトプットの正確性とその時刻の両方に依存するシステム
  - （例）システムへの要求＝「127＋382の答えを求めなさい。答えは3分後までに出しなさい（現在：12時23分）」
    - （答） 509（12時30分）
      ➔ リアルタイムシステムでは計算失敗の例となる
    - （答） 509（12時24分）
      ➔ リアルタイムシステムでも計算成功の例

ソフトリアルタイムとハードリアルタイム

- ソフト・リアルタイム・システム
  - 「ソフトリアルタイムシステム」とは、時間制約を満たす事に失敗した時に、性能低下はあるがシステム故障は起こさない
- ハード・リアルタイム・システム
  - 「ハードリアルタイムシステム」とは、時間制約を満たす事に失敗することが一回でもあると、完全に破壊的なシステム故障につながるかもしれないシステム
- ファーム・リアルタイム・システム
  - ファームリアルタイムシステムとは、いくつかの時間制約ミスではトータルな故障にはならないが、多くの時間制約ミスがあると完全に破壊的なシステム故障につながるかもしれないシステム

！「ソフト」と「ハード」は、求められる時間制約の長短によって区別されるものではない。

リアルタイムシステムの理論モデル（１）
n入力／m出力

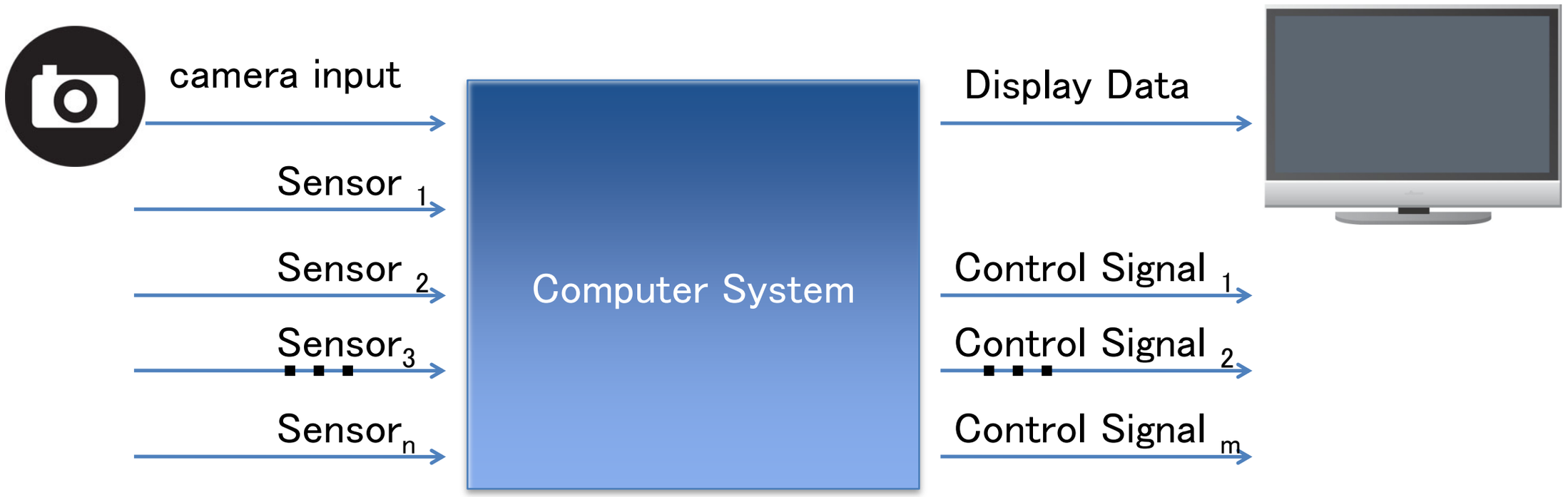


一つの処理の一つの時間制約をだけを満たすのは、簡単（自明）。

リアルタイムシステムは、複数の入力を処理して、複数の出力をする。

複数の時間制約を抱えて、それらをすべて満たすことが要求される。

リアルタイムシステムの理論モデル（２）
リアルタイム制御システム



複数の制御を、それぞれの時間制約を満たして制御することで、
機械のきちんとした動きを実現できる。

時間制約を守るための方法

- 方針1
  - デッドライン（deadline）が近い処理を先に実行する
    - 直感的にも自然な方法
    - 一定の条件のもとでは最適な方法であることが理論的にも証明されている（RMS: Rate Monotonic Scheduling）

- 方針2

  ①各処理それぞれの実行時間の予測

  ②処理時間が予測できれば、すべての時間制約を守るように処理の順番を調整（スケジューリング）

  ③処理時間が予測できない部分は、最悪処理時間を保証できる仕組みを入れる（タイムアウト）

方針１
デッドラインが近い処理を先に実行

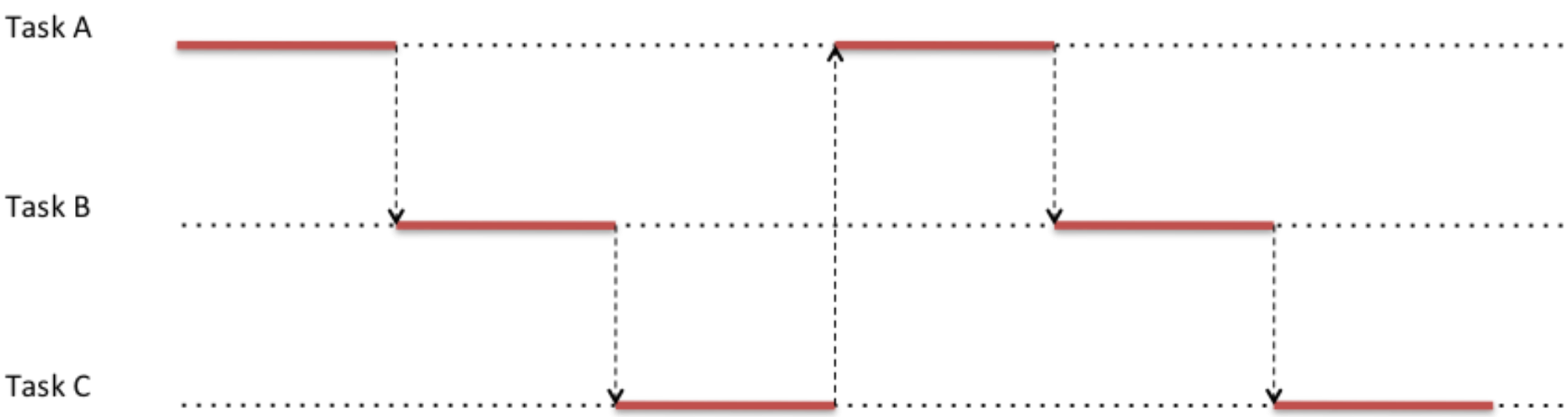
スケジューリング（scheduling）

- 複数の処理があるとき、どういう順番で処理を進めれば、都合が良い・最適な仕事をやれるか、を求める問題のこと。

- リアルタイムシステムでは、時間制約を満たす（＝デッドラインを守る）ことができるような順番で処理をしたい

一般的なスケジューリング

- Round Robin Scheduling（ラウンドロビン）
  - 一定時間毎に処理を順番に実行していくやりかた
  - もともとは大型計算機で、処理のCPU使用時間に比例した従量課金を処理しやすいためのスケジューリング方式
  - デッドラインと処理の順番に関係がなく、リアルタイムシステム向けでない。



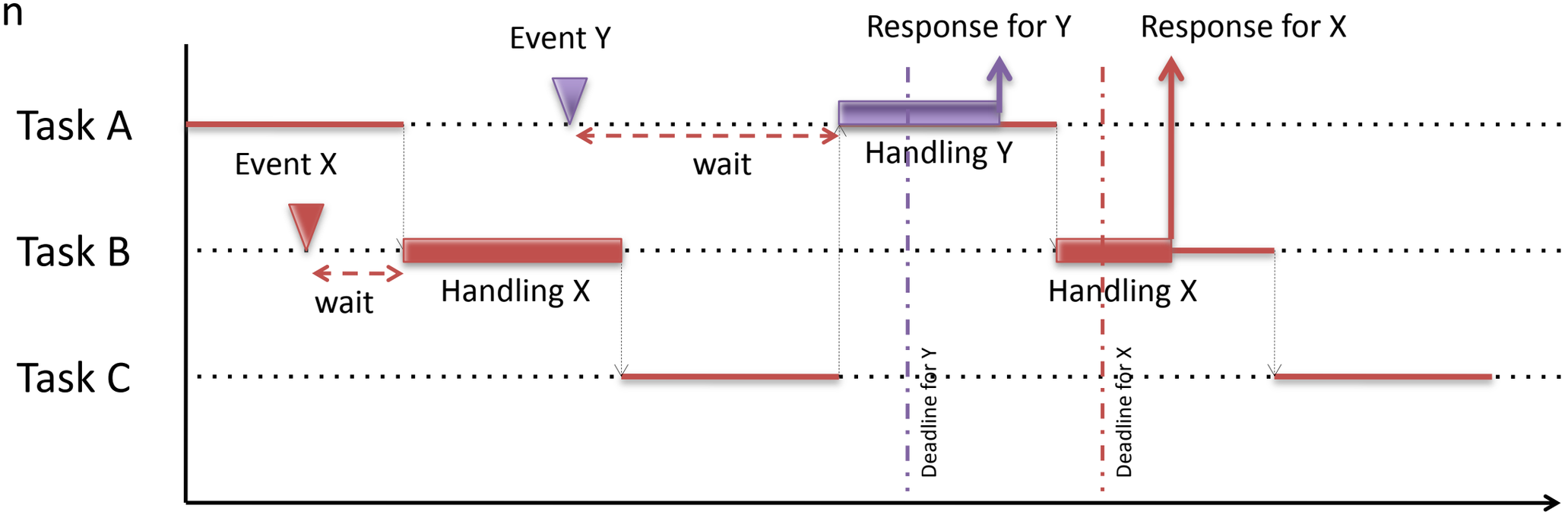
デッドラインが先の処理を先に行なうスケジューリング

- 処理に優先度が付けられる
- 優先度の高い処理は他から（より優先度の低い処理には）邪魔されない。
- 優先度の高い処理は、低い優先度の処理を横取りする（Preemption）

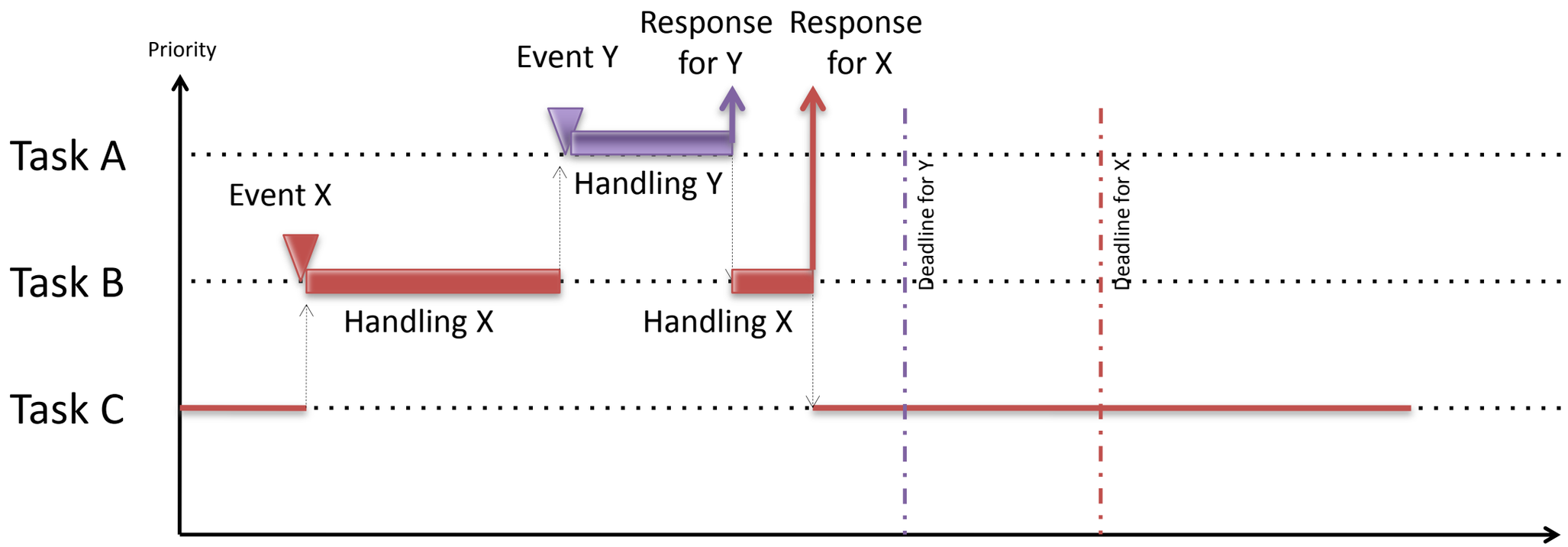
- 優先度の高い処理は、優先度の低い処理より優先して実行されるため、先に終えることができる

より現実に近い例：組込み型と情報処理型の違い

Round Robin Scheduling



Prioritized Preemptive Scheduling



【参考】 Rate Monotonic Scheduling理論

- システムに、処理（タスク）の優先度をつけて、優先度の高い処理は低い処理を横取りできるメカニズムがある前提で、締切が近いタスクから高い優先度をつけて処理するのが、最適スケジューリングであることを証明した理論

- リアルタイムOSが優先度＋横取りスケジューリングを備えている、理論的根拠となっている。

方針２
処理の予測可能化
＋スケジューリング

処理時間を予測できるためには？

- 基本的には、コードを見て、計算機の処理性能がわかれば、処理時間は予測できるはず（理論的には）。

- 予測できないケース1
  - 他の処理に邪魔される場合

  （例）処理の最中に、他の処理に邪魔され、しばらくの間処理が中断していしまうケース。

- 予測できないケース2
  - I/Oやネットワーク通信など、自分が管理できない外部の処理に依存する場合

  （例）インターネットのパケットの返事がいつ戻ってくるか？
    - パケットロスしていたら、永遠に戻らない

  （例）ハードディスクの読み取りがいつ終わるか？
    - 故障していたら、永遠に終わらない

ケース１：他の処理に邪魔される

- （例）他の処理に邪魔されしばらくの間処理ができないケース

- 解決方法
  - 急ぎの処理は優先度を高くして、他の処理に邪魔されない仕組み
    ➔ 処理への「優先度」と「処理の横取り（Preemption）」の導入
  - 優先度の高い処理は、優先度の低い処理を横取りして、優先して実行されるメカニズム

ケース２：自分が管理できない外部の処理に依存

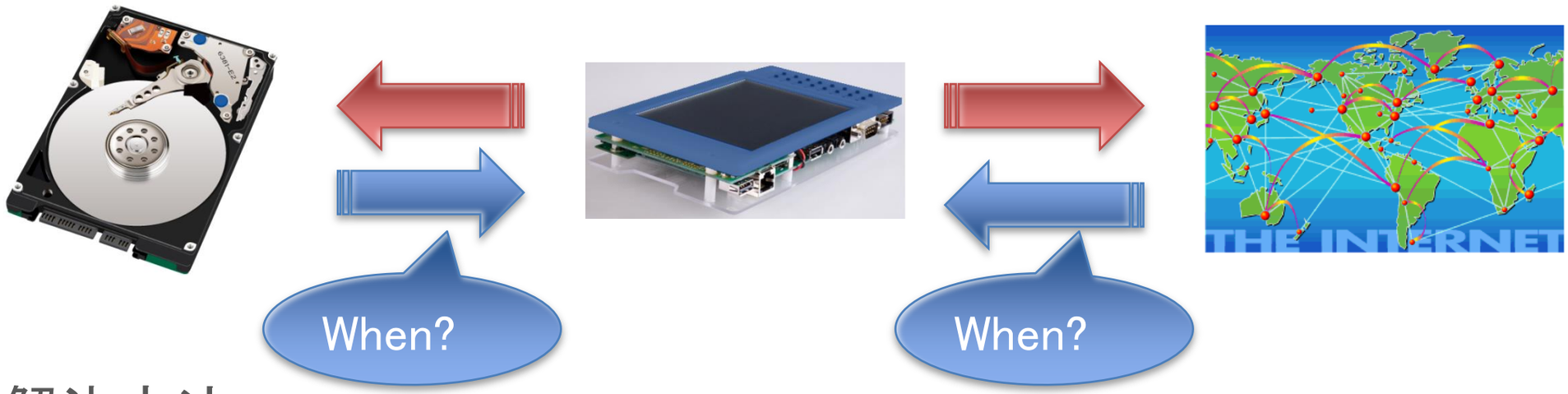
- I/Oやネットワーク通信など

（例）インターネットのパケットの返事がいつ戻ってくるか？

- パケットロスしていたら、永遠に戻らない

（例）ハードディスクの読み取りがいつ終わるか？

- 故障していたら、永遠に終わらない



- 解決方法
  - 処理に“Time Out”を設定し、どうしてもダメな時の最悪時間を設定できるようにする

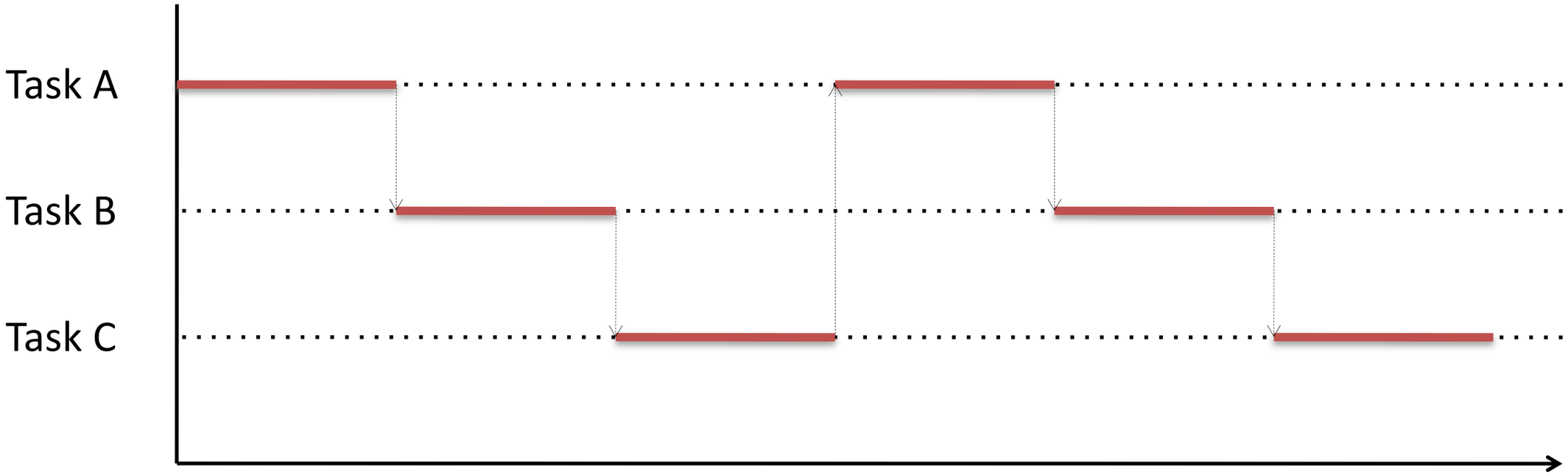
応答性能

高い応答性＝応答性能

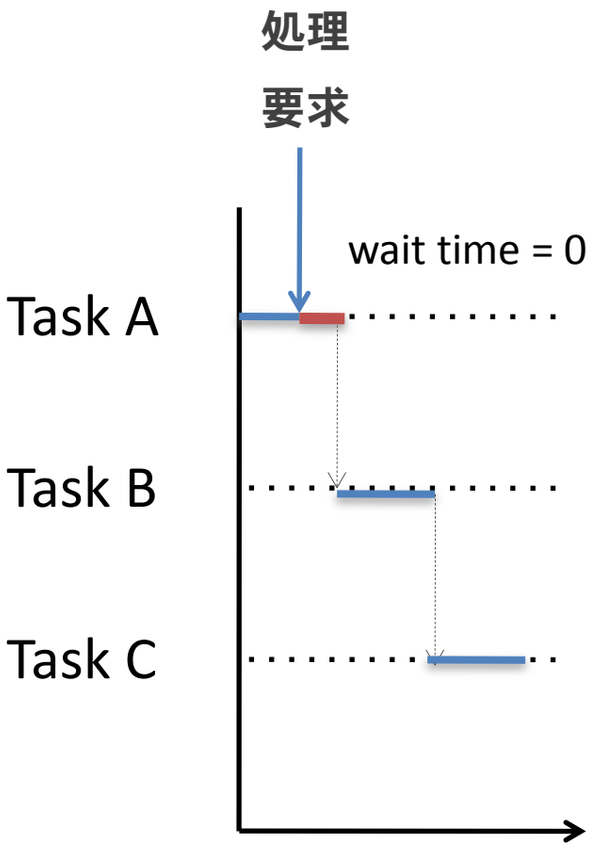
- 応答性能
  - ここでは、「どこまで短い時間制約を満たす能力があるか？」とする。

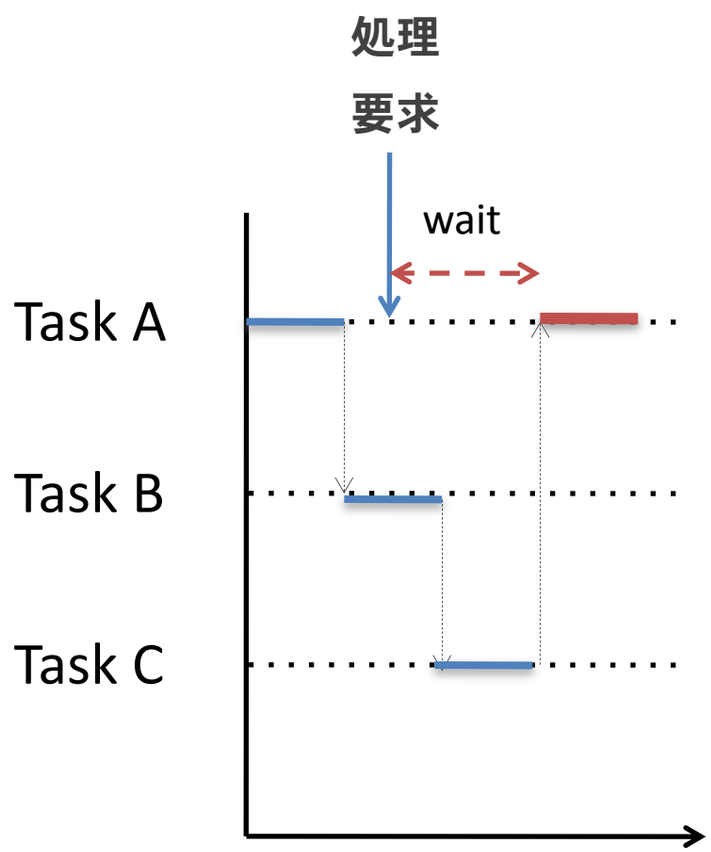
- 考え方
  - 高い性能のCPUを持つ事 → 重要な要件、しかしそれだけではない。
    （次のページ）

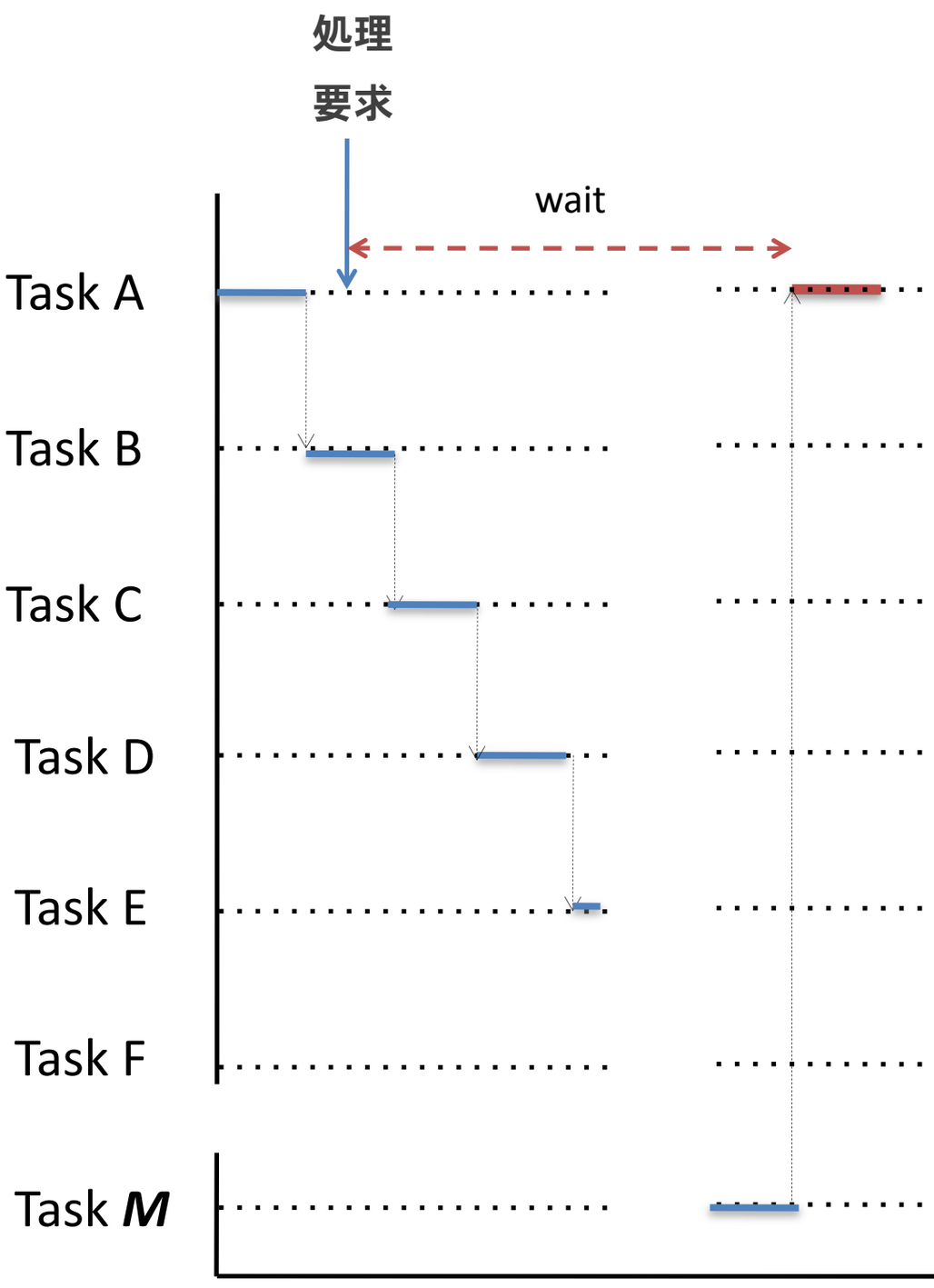
Round Robin Scheduling（ラウンドロビン）



状況とタイミングによってかわる処理実行開始までの時間



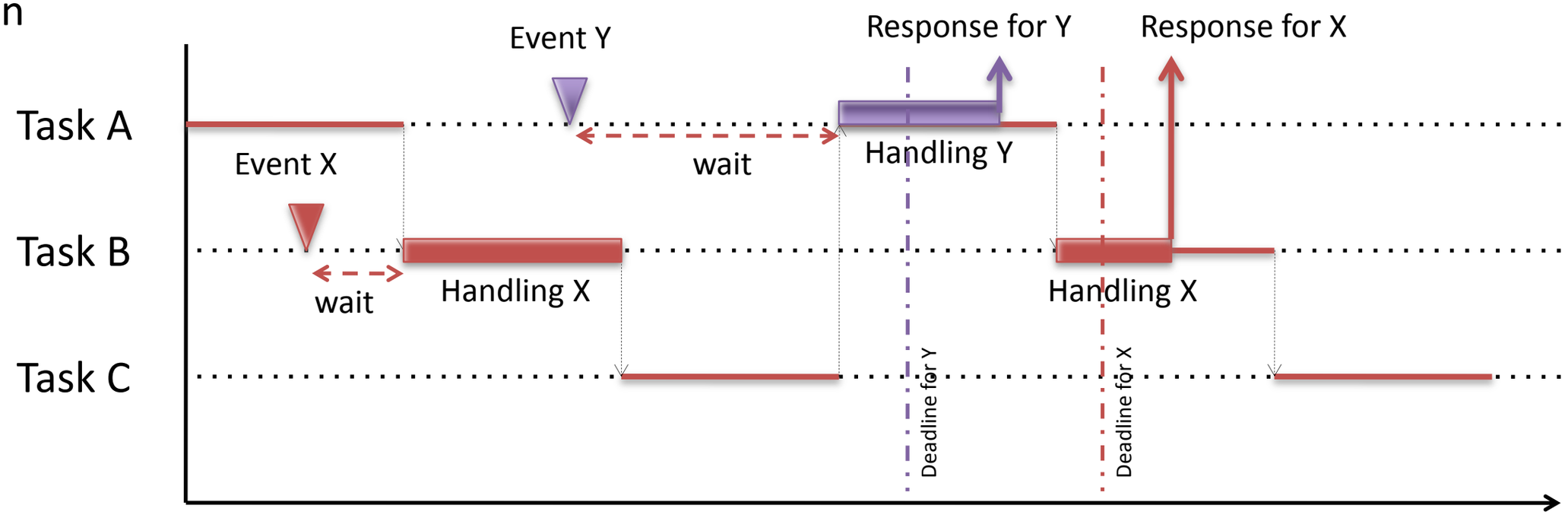




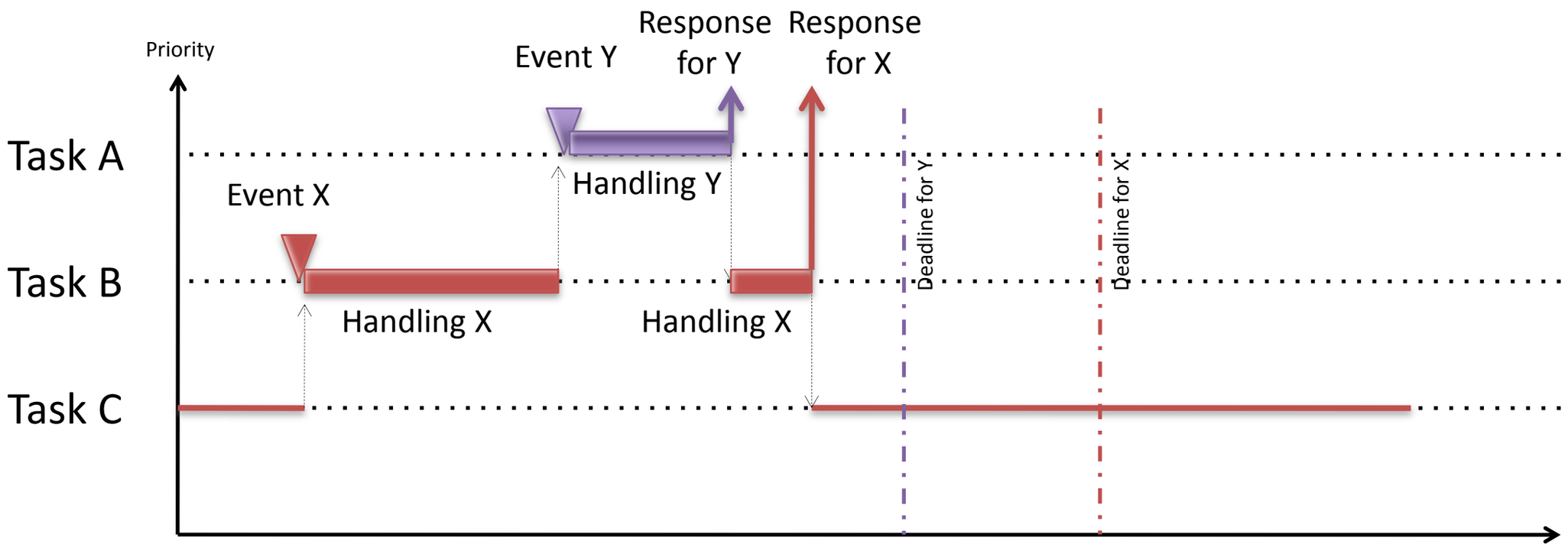
最も短い応答時間と、最も長い応答時間の差を
ジッター（Jitter）と呼ぶ。
OSで保証できる応答時間は、
コンテキストスイッチ時間＋ジッター
になる。

優先度ベースは、応答性能もよい

Round Robin Scheduling



Prioritized Preemptive Scheduling



情報系OSカーネルの応答性能上の限界

- 情報系OSカーネルで行うリアルタイム処理には、技術的限界がある。

- ユーザインタフェースを中心とした、数ミリ秒の時間制約（応答）を扱う場合
  ➔ 情報系OSカーネルでも可能

- 機械制御、通信制御を目的とした、マイクロ秒を争う時間制約（応答）を扱う場合
  ➔ 実現のためには、リアルタイムカーネルが必須

まとめ

- リアルタイムシステム
  - 故障のような厳しい結果をもたらす応答時間制約を満たすことができるシステム

- 締切が近い処理を優先的に実行
  - 優先度ベース、Preemptive（横取り）スケジューリング
  - Rate Monotonic Scheduling
- 処理の予測可能性に基づいたスケジューリング
  - 高い優先度づけによって他のタスクに邪魔されない
  - タイムアウト機能による最悪時間保証
- 高い応答性能（＝短い応答時間制約が扱える） = Jitter性能
  - 優先度ベース、Preemptiveなスケジューリングが有効

第４章
組込みリアルタイムシステムの機能

組込みリアルタイムシステムの例

[EVENT]
コピーボタン押下

Copy Processing Task

[EVENT]
印刷データ受信

Print Processing Task

[EVENT]
コピー文書セット

Copy Processing Task

[EVENT]
紙づまり

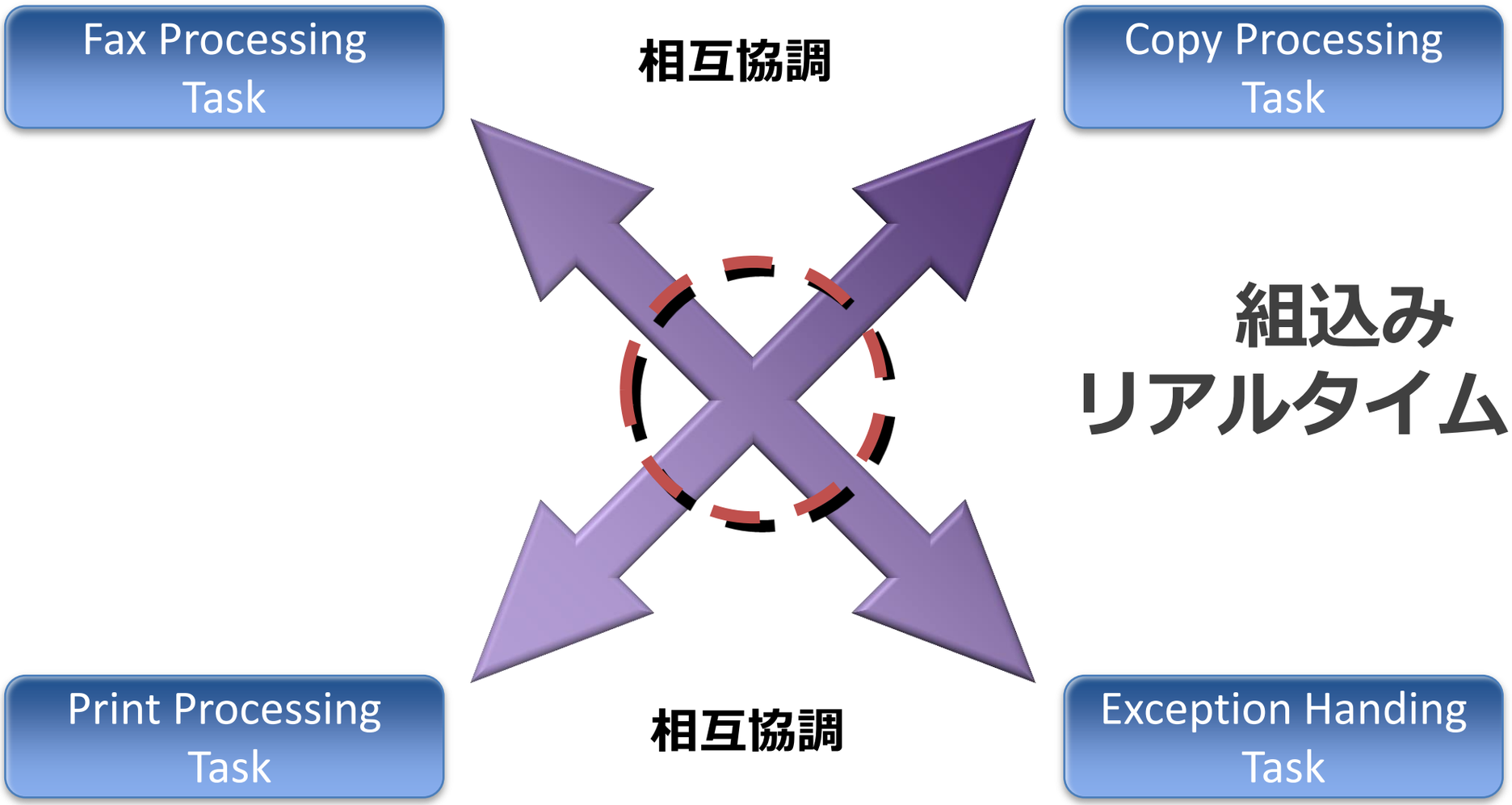
Exception Handing Task

[EVENT]
FAX受信

Fax Processing Task

all-in-one (copier, fax, and printer) machine

組込みリアルタイムシステムの内部



組込みリアルタイムOSの機能

- 複数のタスクやハンドラの間の協調動作を管理する

- 具体的には…

1. タスク（スレッド、プロセス）管理
   - スケジューリング、ディスパッチング
2. タスク間同期
3. タスク間通信
4. 資源（記憶）管理
5. 時刻／時間管理

1 タスク管理
Task Management

組込みリアルタイムシステムの例（再度）

[EVENT]
コピーボタン押下

Copy Processing Task

[EVENT]
印刷データ受信

Print Processing Task

[EVENT]
コピー文書セット

Copy Processing Task

[EVENT]
紙づまり

Exception Handing Task

[EVENT]
FAX受信

Fax Processing Task

all-in-one (copier, fax, and printer) machine

マルチタスク処理

- 並列に動作する独立な処理＝タスク（task）
  - …各「タスク」が独立に発生した事象を扱う
  - ファックス受信 ➔ Fax Processing Task
  - 印刷データ受信 ➔ Print Processing Task
  - コピーボタン押下 ➔ Copy Processing Task
  - 紙づまり検知 ➔ Exception Handling Task

- 1つのコンピュータの上で、複数の独立した処理を同時に動かす機能 ➔ マルチタスク処理（multitask）

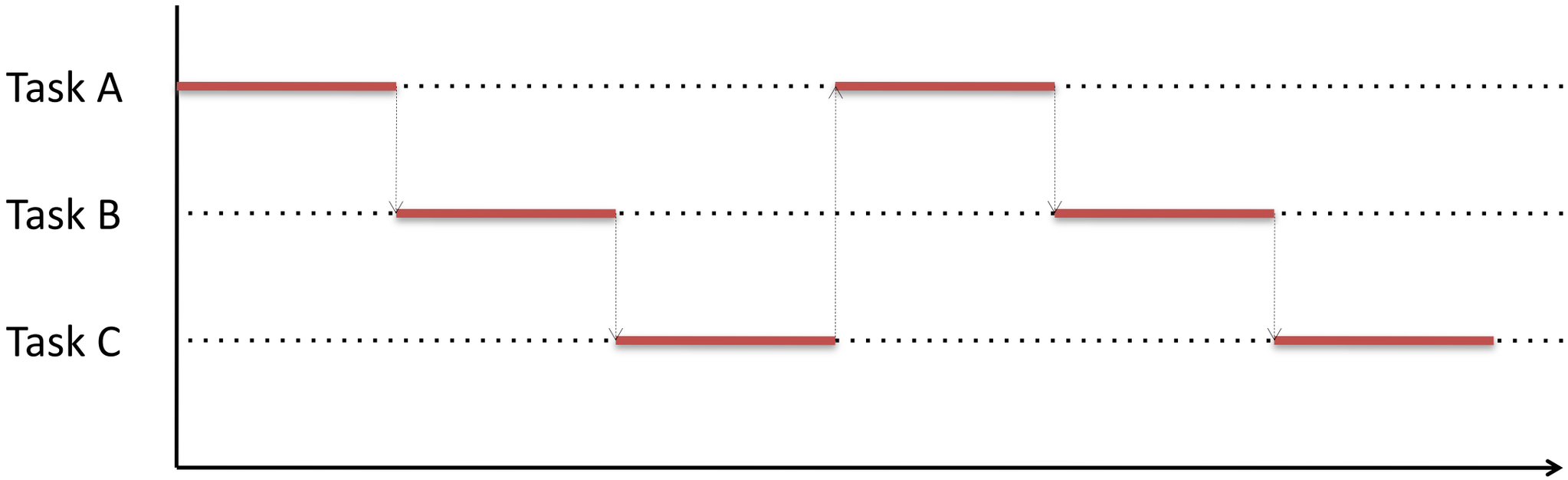
タスクスケジューリング（task scheduling）

- 一つのコンピュータ上で、複数のタスクに動作させる処理＝マルチタスク処理
- 同時とはいっても、CPUを使う時間帯を複数のタスクに別々に振り分けて、順番に使う
  ＝タスクスケジューリング

  ※時間分割の長さ、分割方法によって様々な方式

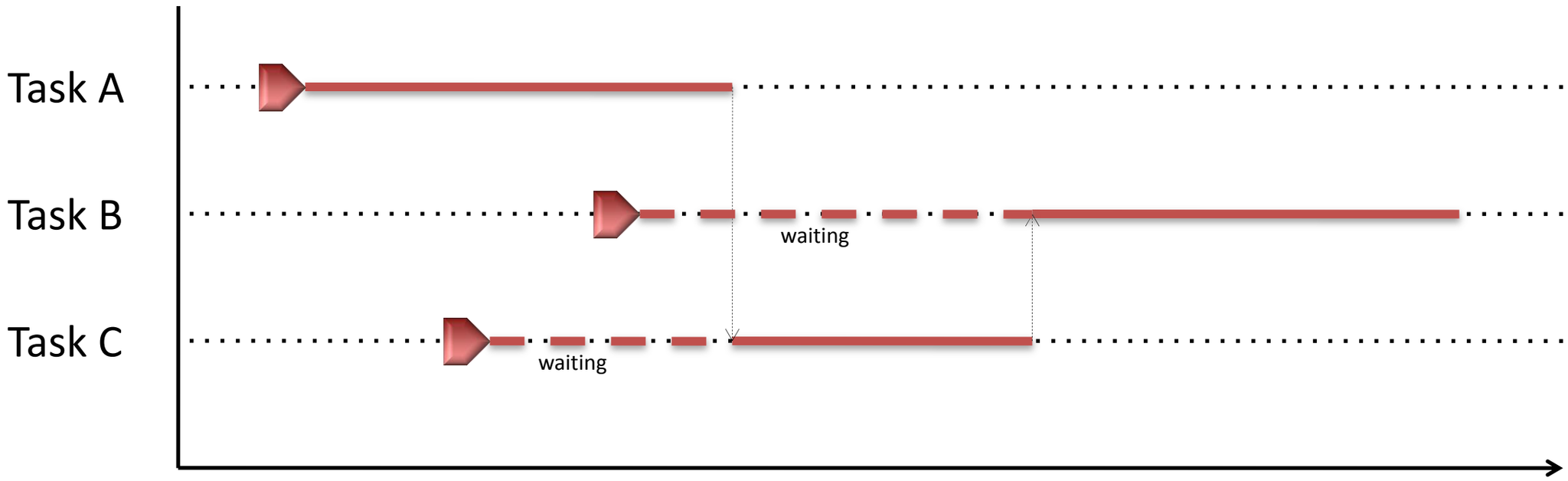
ラウンドロビン方式（Round Robin Scheduling）

- ラウンドロビン方式は、OSによる最も簡素なスケジューリング方式
- 各タスクに同じ時間だけ同じ順番で、優先度をつけずにスケジューリングする
- 簡素で実装も容易である。また、タスクはstarvationが発生しない。



FCFS方式 (First Come First Served Scheduling)

- FCFSは待ち行列に到着した順番に従ってスケジュールされる方式
- 優先度をもたず、時間制約（デッドライン）のある処理には適さない



リアルタイム方式（Real-time Scheduling）

- 以下の方式を融合したものが一般的

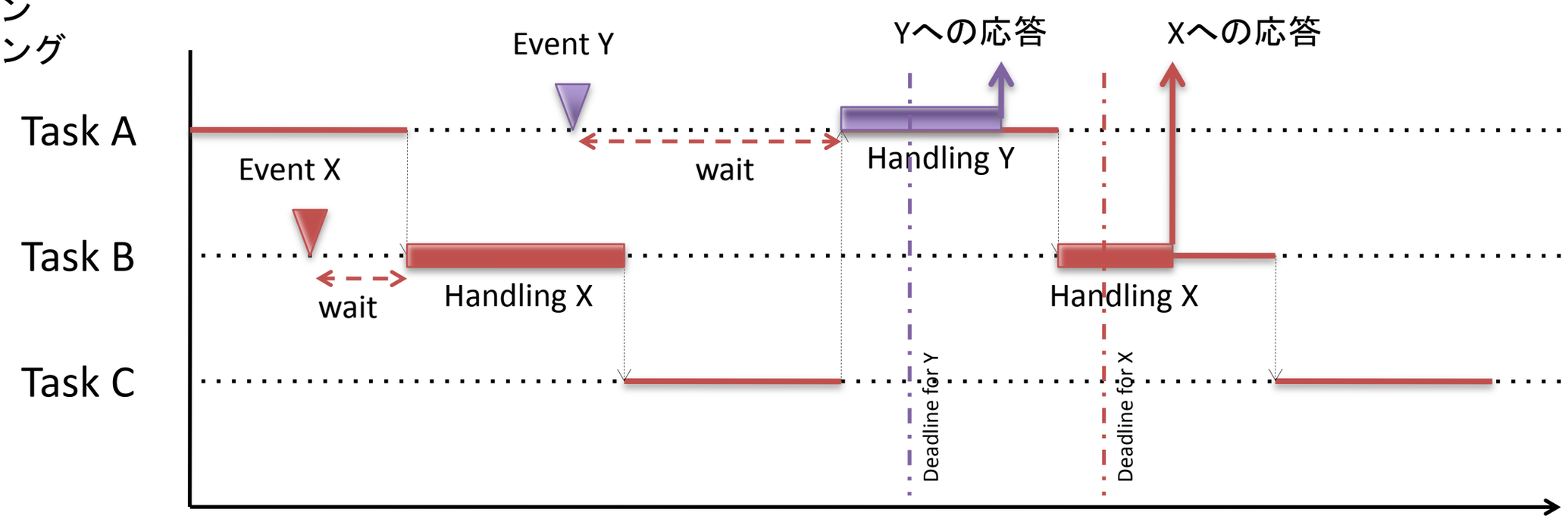
- 優先度ベース（Priority-Based Scheduling）
  - タスク（Task）に優先度（Priority）を持たせることができる.
  - 高い優先度のタスクは、低い優先度のタスクより先にCPUの割当を受けることができる。

- 横取り型（Preemptive Scheduling）
  - CPUを他のタスクに割り当てることを決めたときに、CPUから強制的にタスクを取り除くことができる。

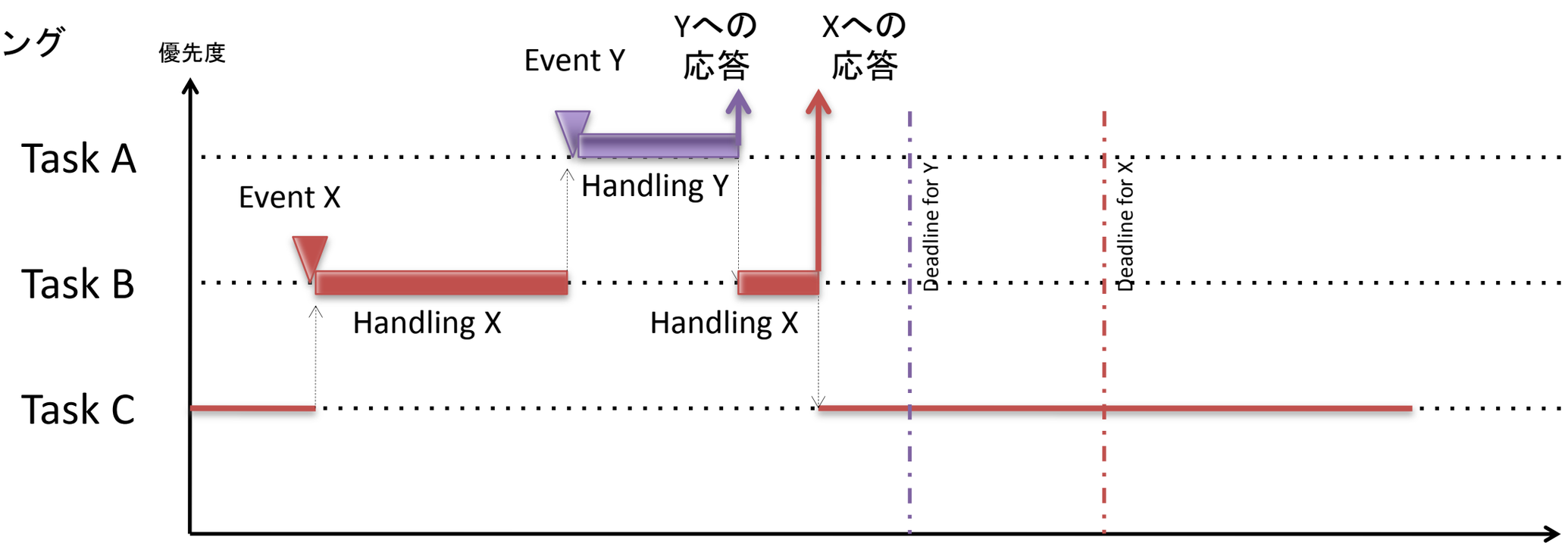
- イベント駆動型（Event-Driven Scheduling）
  - 高い優先度のイベントへのサービス要求が発生した時のみ、タスクスイッチが起る。
    - これは、“preemptive priority”または “priority scheduling”とも呼ばれる。

優先度ベースとラウンドロビン

ラウンドロビン
スケジューリング



優先度ベース
スケジューリング



2　同期
Synchronization

同期（Synchronization）の必要性

- タスクの相互関係
  ➔ タスクが全く独立なわけではない。
  ➔ 互いに関係がある…どんな関係？

- 【相互関係1】仕事の依存関係
  - Aの仕事がおわらないと、Bの仕事が始まらない。
    （例）学生のレポートが提出されて、初めて教師は採点する。
- 【相互関係2】道具を共有している関係
  - AとBが同じ道具を使って仕事をする。
    （例）2人の大工に1つののこぎり

- タスクの間の相互関係に合わせて、処理の実行を調節すること →
  同期（Synchronization）

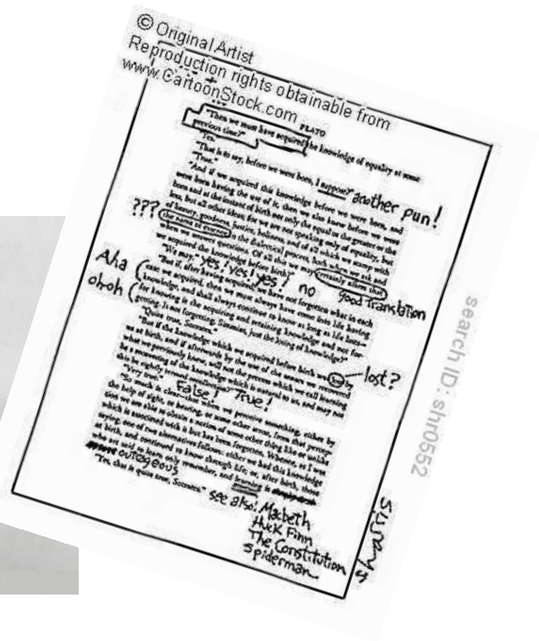
【相互関係１】仕事の依存関係による同期

- 依存関係
  - Aの仕事がおわらないと、Bの仕事が始まらない。
    （例）学生のレポートが提出されて、初めて教師は採点する。

- 必要な同期機構
  - 生徒Aの宿題が終わる→（同期）→教師Bが採点を開始する

生徒A

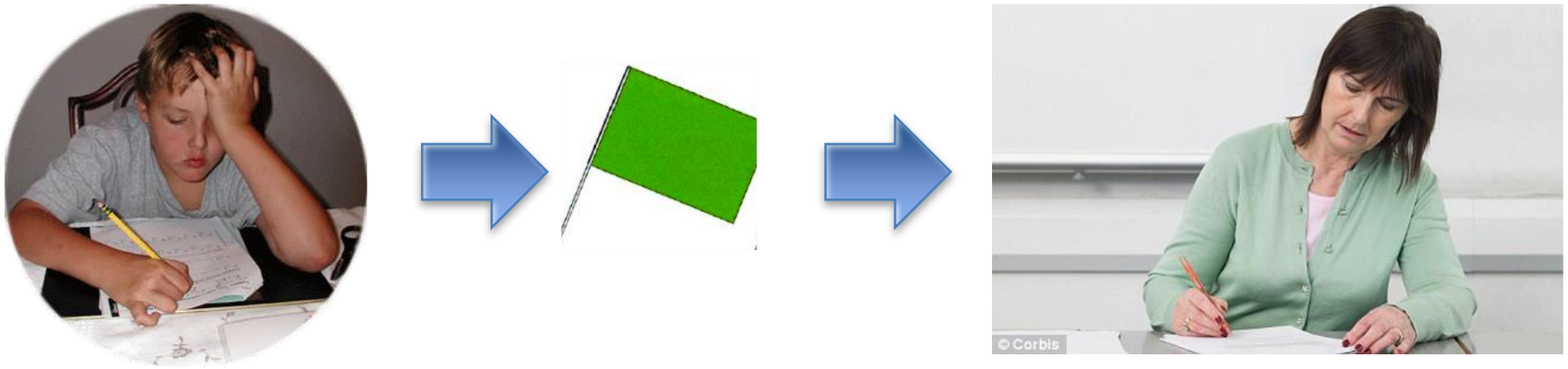
宿題



教師B

【相互関係１】仕事の依存関係がある時の同期
イベントフラグ（event flag）

- 仕事に依存関係がある状況で…
  - 先に仕事をしていたタスクが、ある処理が終わったという「事象（イベント）」を、次のタスクに伝える → イベントフラグ



- イベントフラグの機能
  - 事象の通知をタスク間で通信する
  - 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

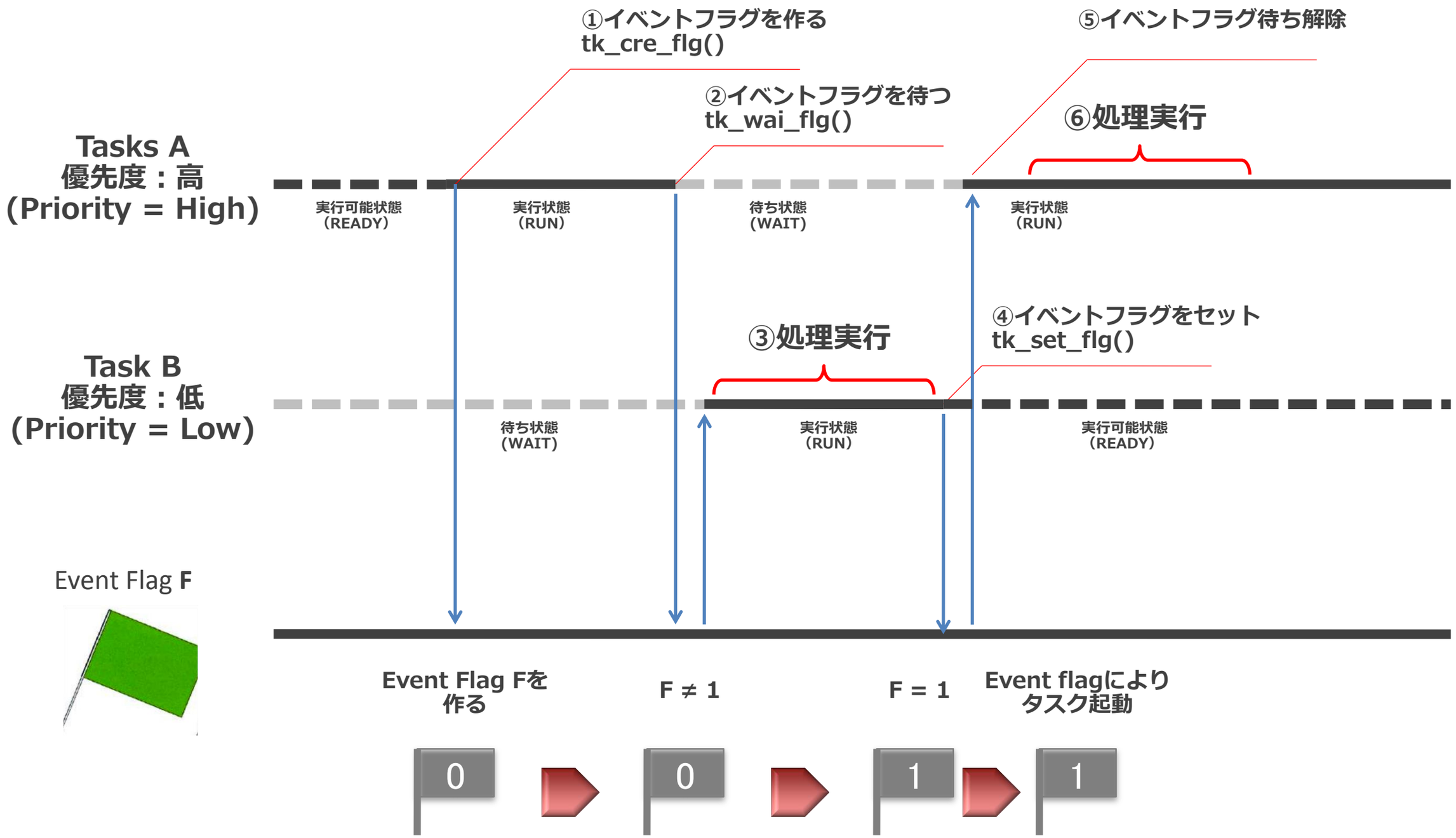
イベントフラグの基本機能

- イベントフラグは、2種類の状態を持つ
  - Set状態
  - Clear状態

- イベントフラグへの基本操作
  - セット (Set)命令 → イベントフラグは”Set”状態になる
  - クリア (Clear)命令→ イベントフラグは “Clear”状態になる
  - ウェイト (Wait)命令
    - イベントフラグが”Clear”状態
      → そのタスクをイベントフラグが”Set”状態になるまで待たせる。
    - イベントフラグが”Set”状態
      → そのタスクは動作をそのまま継続する

- 1つのイベントフラグは1ビットで実現可能
  - 上記の基本操作では8個、16個等のイベントフラグをまとめて操作
  - 複数イベントフラグのAND待ちやOR待ちができる

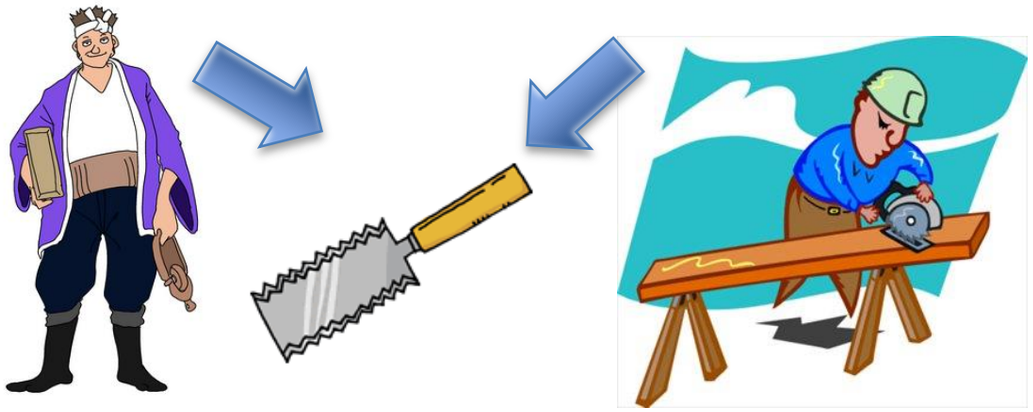
イベントフラグの例：Task B終了→Task A起動



【相互関係２】道具の共有関係による同期

- 依存関係
  - AとBが同じ道具を使って仕事をする。
    - （例）2人の大工に1つののこぎり

- 必要な同期機構
  - Aがのこぎりを使い終わる→（同期）→Bがのこぎりを使い始める

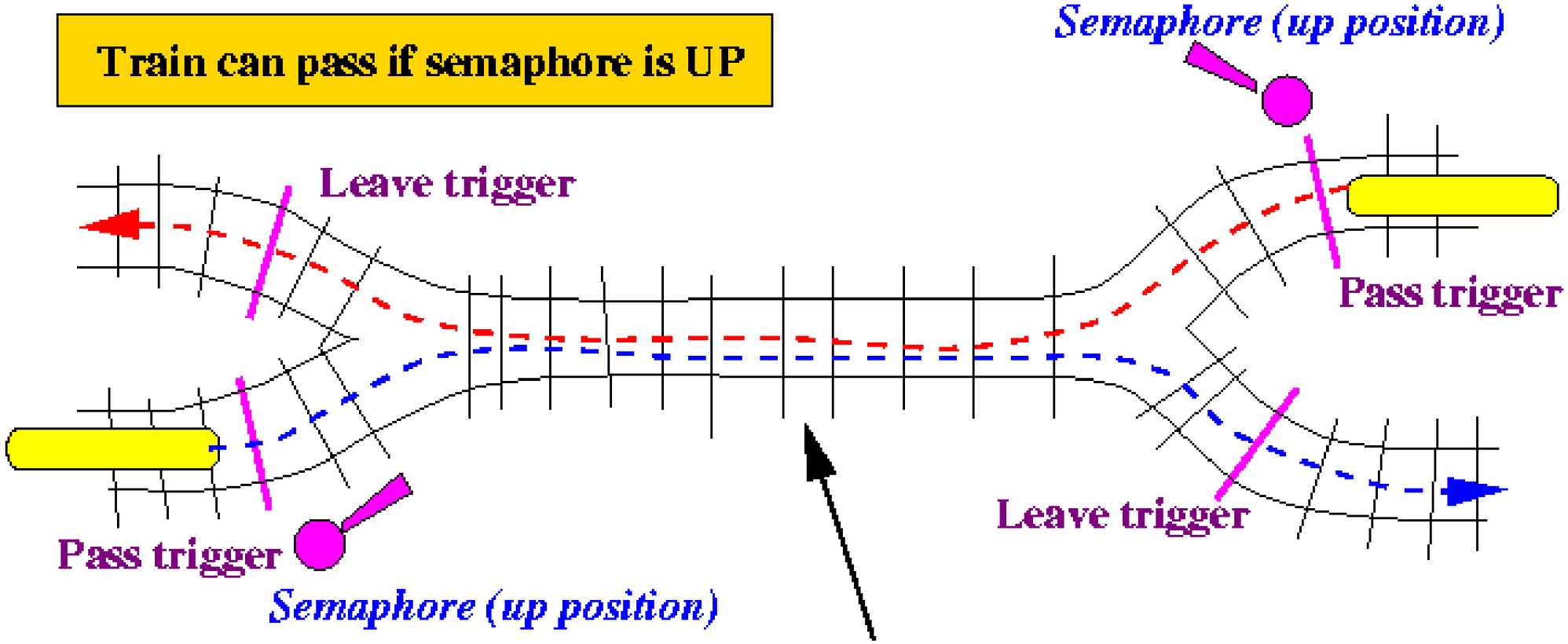


【相互関係２】道具を共有している時の同期 排他制御（mutual exclusion）

- 排他制御（mutual exclusion）
  - 同期方式の一つ
  - 複数のタスクで、道具を共有している関係で、同じ道具を複数のタスクが同時に使うと、おかしなことがおこる。そこで、ある道具を使えるタスクを一つに限定し、その処理が終わるまで、他を排除する制御が必要

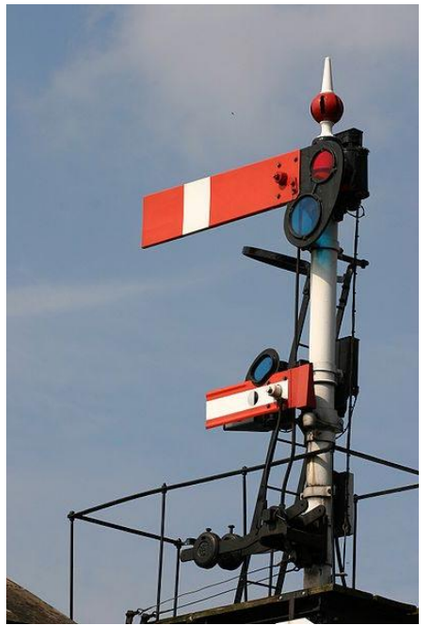
- 実例
  - データを更新している最中にそのデータを読み出すと、中途半端な値が得られてしまう。
  - データの更新処理が始まってから終わるまでは、同じタスクがずっとアクセスし続けるようにする。

セマフォア（Semaphore）（1）



セマフォア（Semaphore）（2）

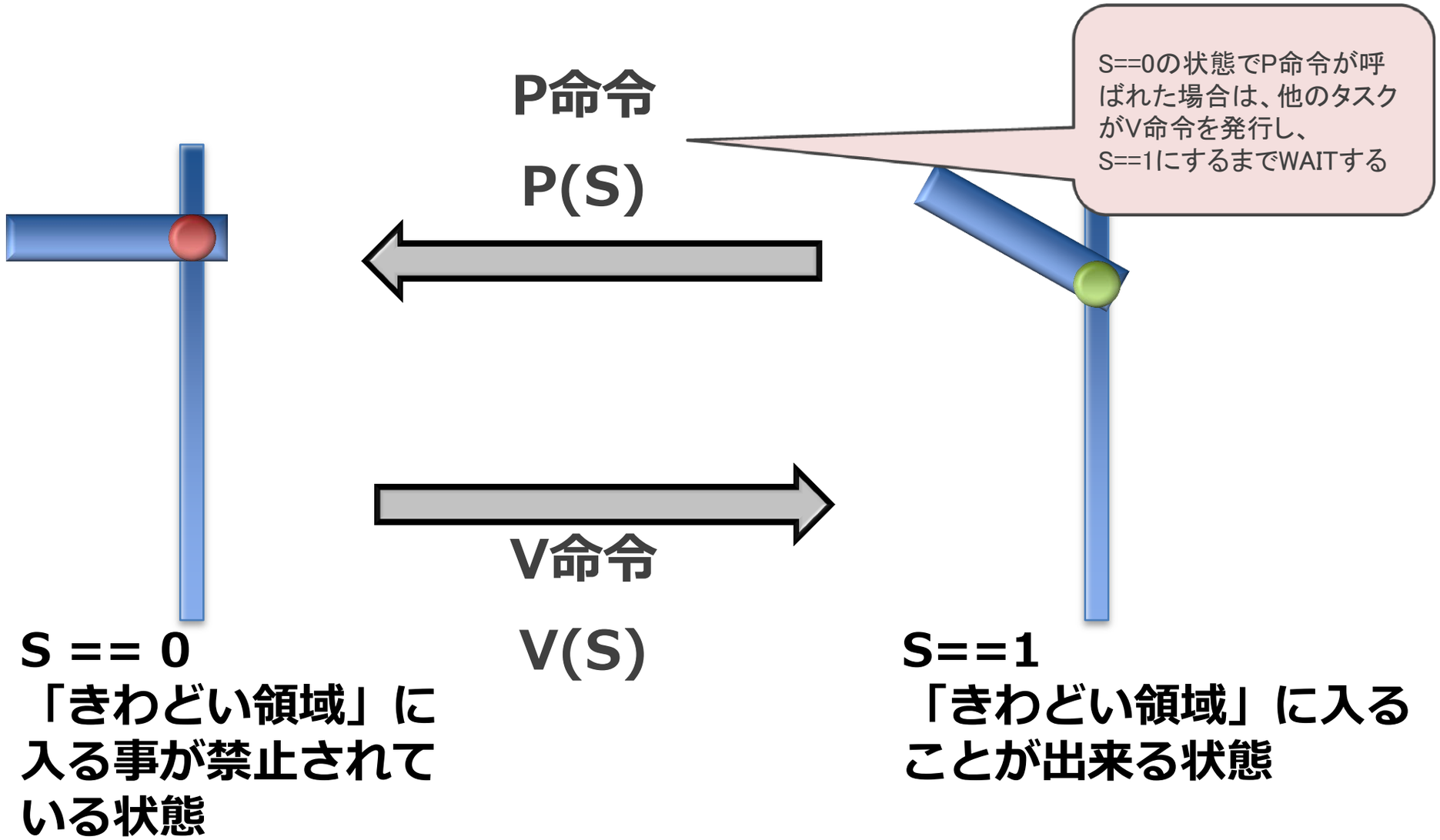
- 並行プログラミング環境において、複数のタスク(プロセス)から共通資源へのアクセス を制御するために用いられる、防御変数 or 抽象データ型



- セマフォは競合状態を防ぐ為に用いる。

- 問題点
  - セマフォアを用いると、デッドロックが起こる可能性がある。
    - 例 : dining philosophers problem

セマフォアの基本動作（バイナリセマフォア）



T-KernelでのP命令、V命令

一般機能名	T-Kernel システムコール
P (S) wait semaphore	tk_wai_sem()
V (S) signal semaphore	tk_sig_sem()

２種類のセマフォア

- バイナリセマフォア
  - 共有資源にアクセスするタスク(プロセス)を一つに限定するためのメカニズムで。
  - 変数の値として“true/false” (= locked/unlocked) を持つ。

- 計数セマフォア
  - 共有資源にアクセスするタスク(プロセス)の数を、決まった複数個に限定するメカニズム。
  - アクセスする上限数～0の間の値をとる。

競合状態の回避するプログラミング例

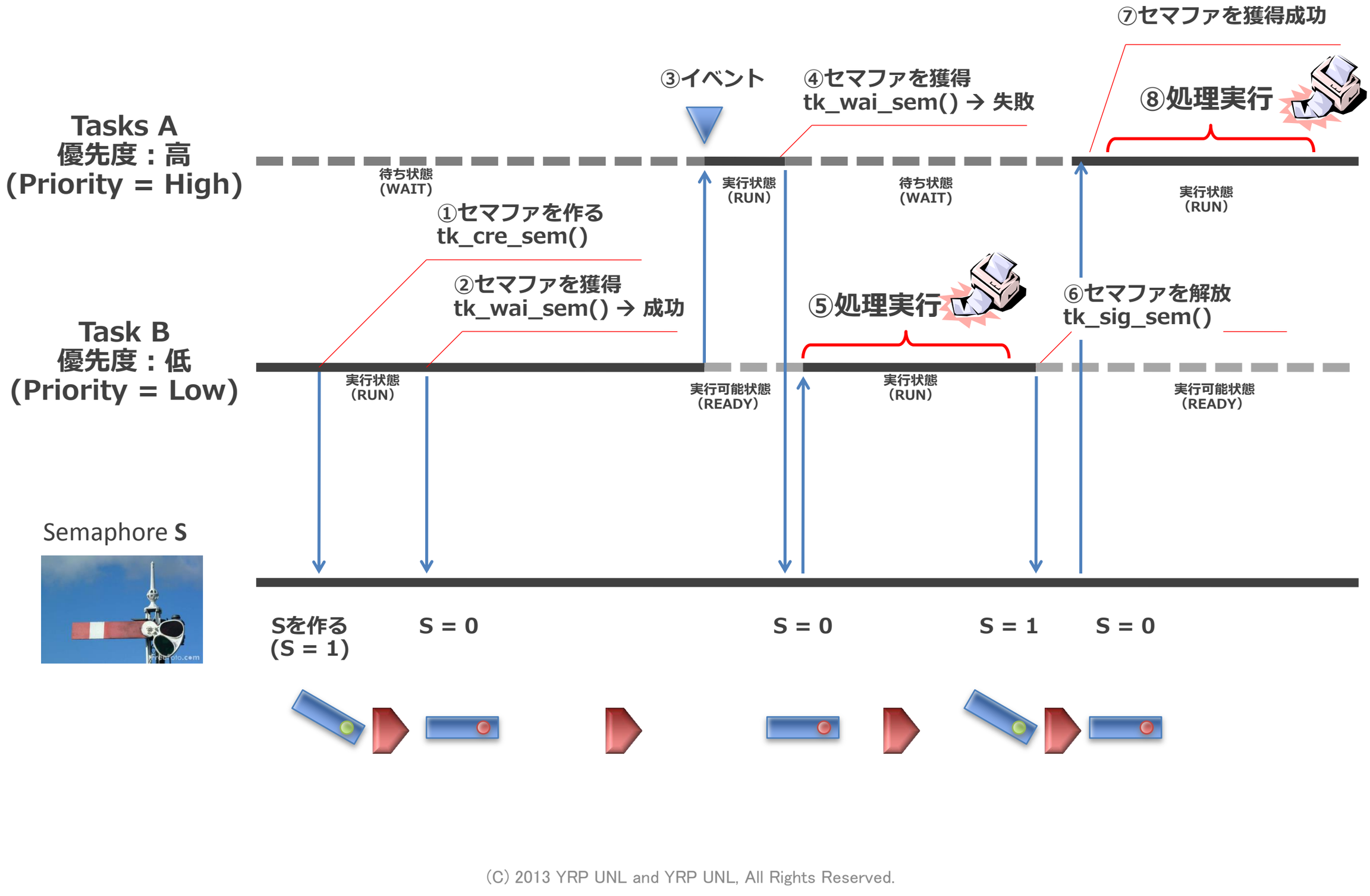
Task 1

```
:
P(S);
…際どい領域
   （競合しうる状態）…
V(S)
:
```

Task 2

```
:
P(S);
…際どい領域
   （競合しうる状態）…
V(S)
:
```

セマファの例：Task B獲得→Task A獲得



3　通信
Communication

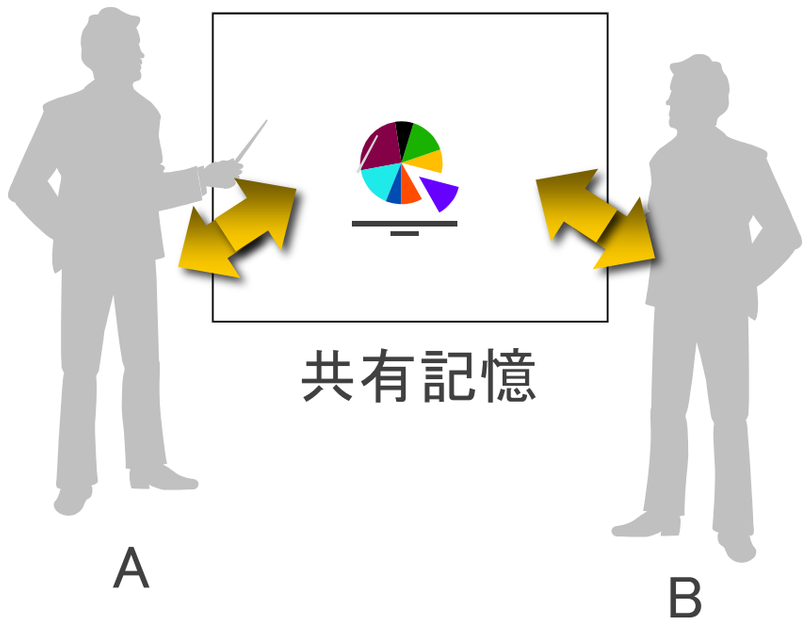
通信　（Communication）

- 仕事・処理の結果や情報を、あるタスクから別のタスクに伝えてやること＝通信（Communication）

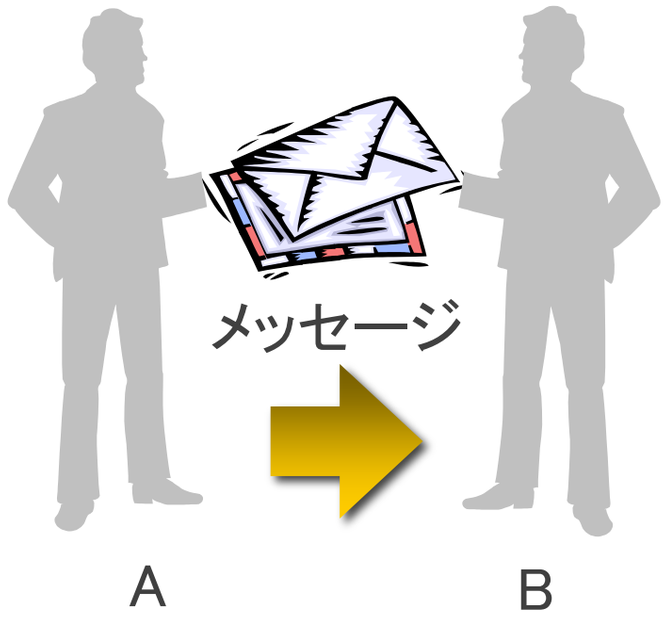
- 例：学生がレポートを書く➔ 教師はレポート採点
  - 学生・教師間の通信 ＝ レポートの内容

通信の方式（１）
共有記憶とメッセージ

- 複数のタスクが通信する方式の分類...
- 共有記憶方式とメッセージ方式
  - 共有記憶方式（Shared Memory）
    - AとBが同じ記憶領域を読み書きして情報交換
  - メッセージ方式（Message Passing）
    - AからBへ情報を渡して情報交換



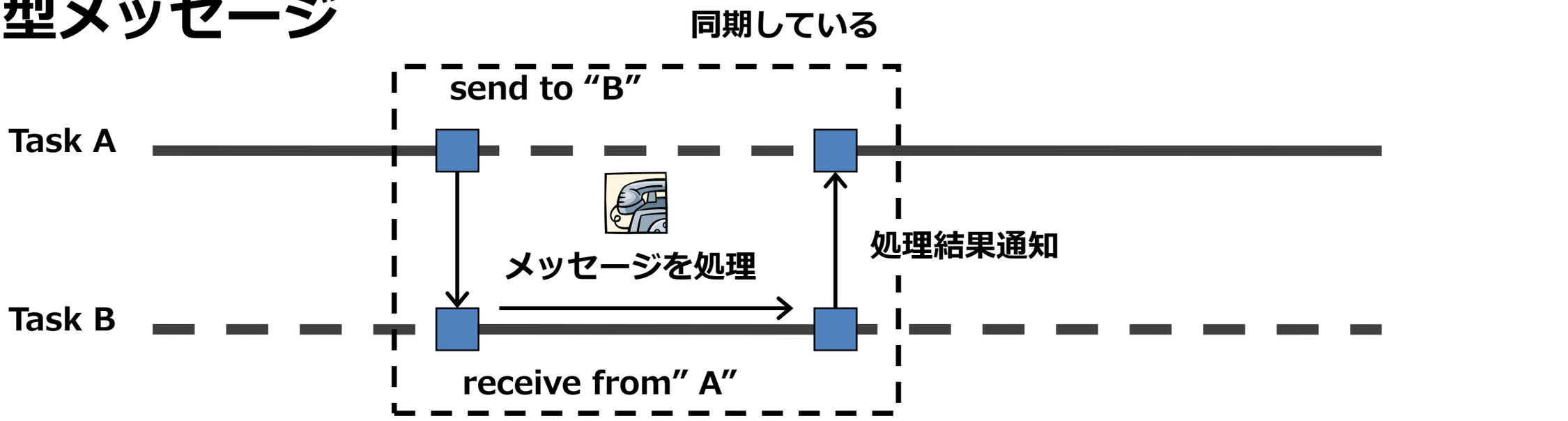
共有記憶方式



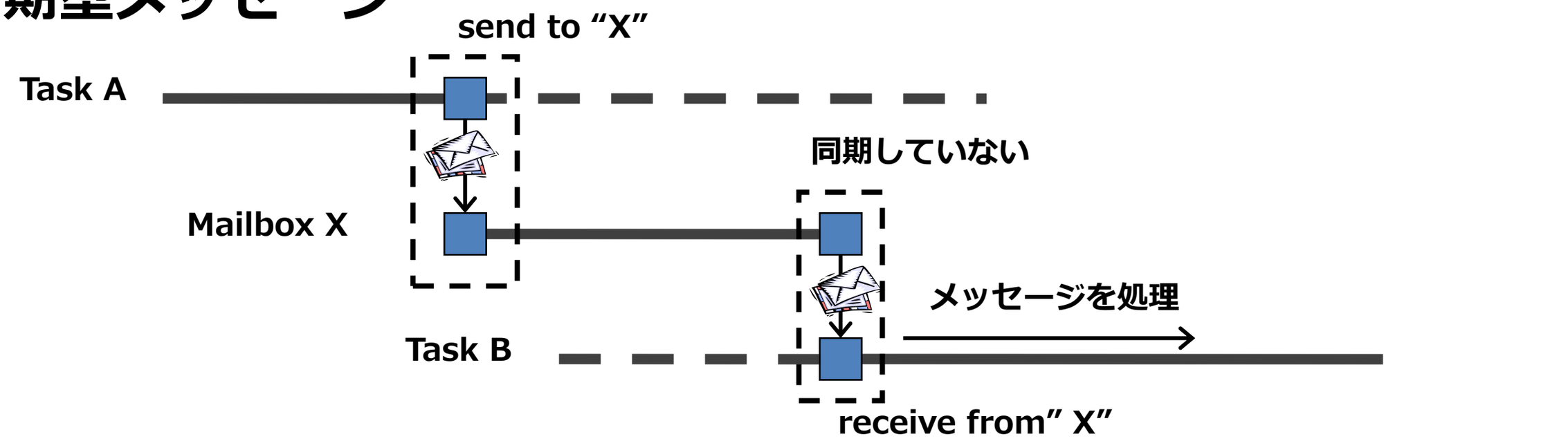
メッセージ方式

通信の方式（２）
同期型メッセージ・非同期型メッセージ

同期型メッセージ

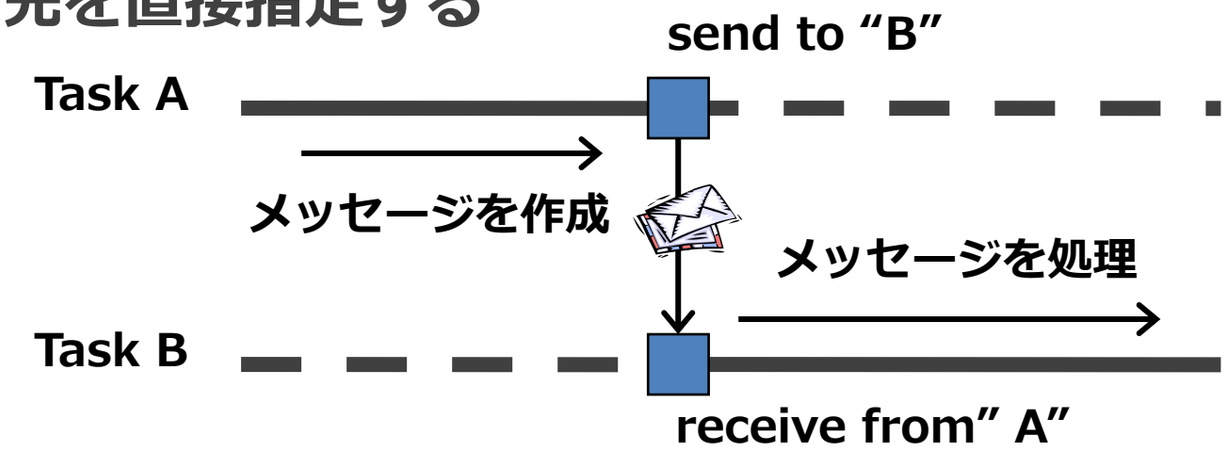


非同期型メッセージ

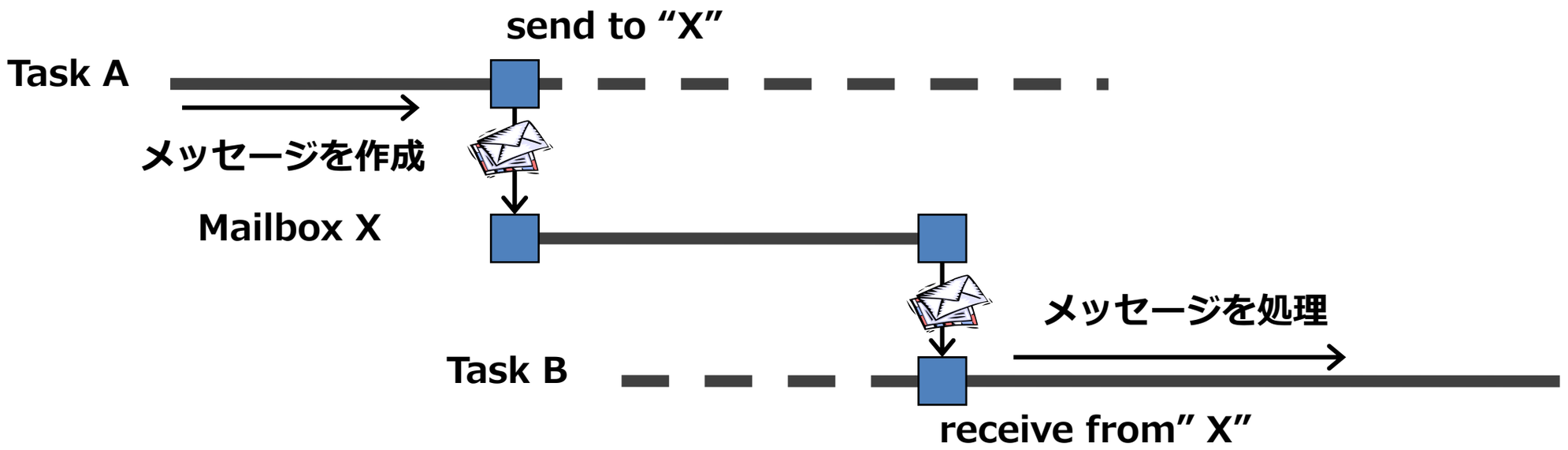


通信の方式（３）
直接通信・間接通信

直接通信
送信先を直接指定する



間接通信
通信の媒介となる資源を指定する



（補足）直接通信、間接通信の例

- 直接通信
  - タスクイベントの送信
  - tk_sig_tev (ID tskid, INT tskevt);

- 間接通信
  - メッセージバッファへ送信
  - tk_snd_mbf (ID mbfid, VP msg, INT msgsz, TMO tmout);

同期と通信

- 同期と通信は同時に起こることが多い

- 例
  - 仕事が終わる
  - 仕事の内容を伝えると同時に、処理を切り替える。

- 同期をするためには、通信しなければならない
  - 少なくとも、仕事が終わったという、「1ビット」の情報は通信する必要

4　実時間処理
Realtime Processing

時間を扱う機能

- リアルタイムシステム／実時間システム

- 時間を扱ったプログラムを書くことが必須

例えば、.....

- 5分たったら休みたい。
- 7時になったら起こしてね。
- 5分おきに○×したい。
- ところで、今何時？

時間を明示的に扱う機能

- 時刻の取得、設定
  - 相対時間
  - 絶対時間
- 時間割込み処理
  - ある時間になったら、あらかじめ決めておいたモジュールのルーチンを起動する
  - アラーム
  - 周期割込み
- タイムアウト指定
  - ある一定時間以内に処理が終わらない⇒その処理をやめる。
  - タイムアウト指定つきサービスコール

１．時刻の取得、設定

- 相対時刻
  - OS立ち上げ時からの相対的な時刻。
  - ハードクロックのカウンタを持つハードウェアタイマを使用する。
- 絶対時刻
  - 年、月、日、時、分、秒で表される一意の時刻。
  - 不揮発の時刻情報を保持可能なRTC(Real Time Clock)のハードウェアを使用する。

２．タイマ割込み処理

- アラーム
  - 特定の時刻または一定時間後にタイマ割込みを発生させ、特定の処理を実行させる。
  - 例:ビデオ録画予約では、予約した日時にタイマ割込みが発生し、録画処理が実行される。
- 周期割込み
  - 一定時間間隔でタイマ割込みを発生させ、処理を実行させる。
  - 例:ビデオ表示では、33m秒に1回割込みを発生させ、割込み毎に1フレームの表示を行う。

3．タイムアウト処理

- 待ち時間が永久に継続される可能性がある場合
  - 予め待ち時間を設定し、待ち時間が経過した場合、別の処理を実行させたい。
  - 例：ハードディスクのI/O完了待ち
  - ディスクの故障時にI/Oが完了しない場合がある。このため、I/O完了待ち時間を設定し、その時間が経過した場合、ディスク故障のメッセージを表示する等のエラー処理を行う。

5 記憶管理

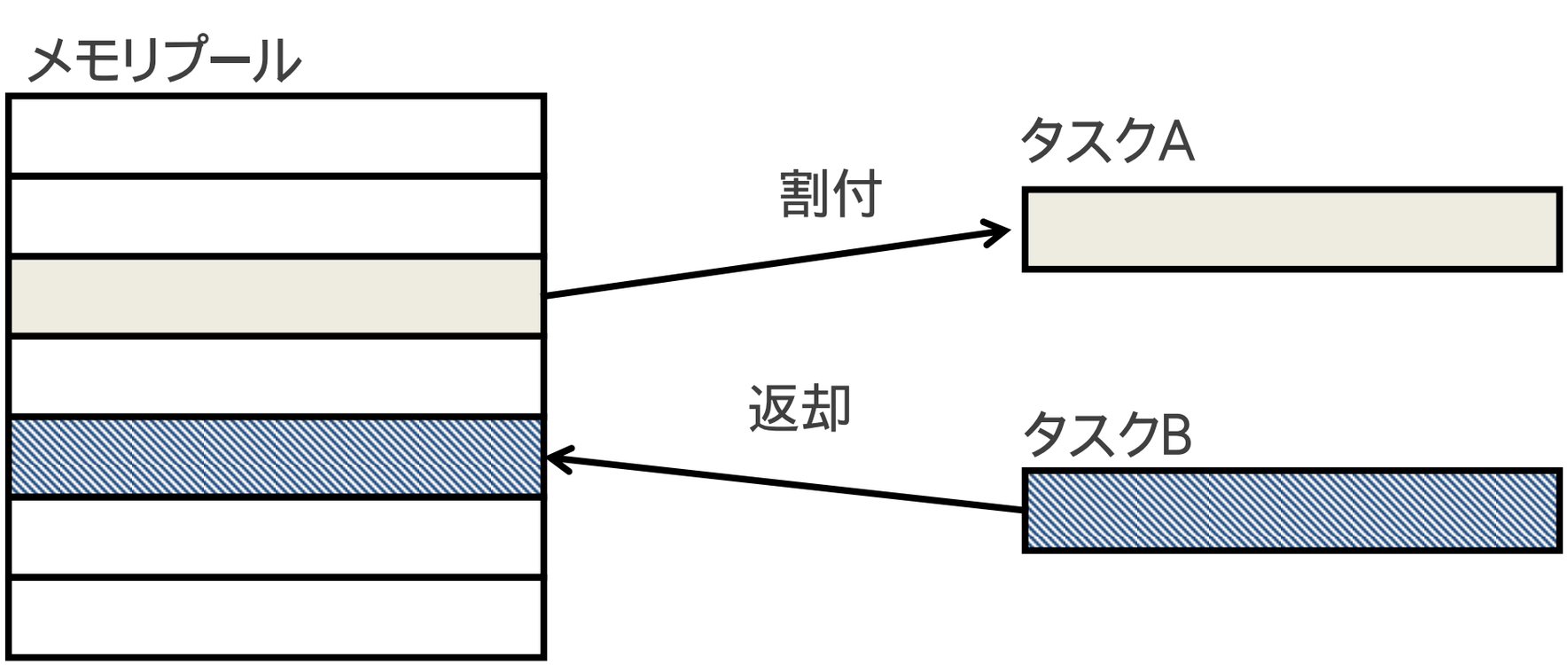
メモリプール管理（１）

- 処理の途中で多くのメモリが必要になること
  - 必要最大のメモリを常に独占するのは無駄
  - 必要な時だけにメモリを確保する
  - 必要がなくなったら解放する

  そのためには.....
- どのメモリをどのタスクが使っているか管理
  ➔メモリプール管理機能

メモリプール管理（２）

- あらかじめ確保した大きなメモリ領域を管理
  - タスクからの要求に応じて、空いている部分から必要量だけ割り付ける（memory allocation）
  - 不要になったメモリは返却（memory free）



【発展】
メモリマップI/O

- I/Oもメモリ空間に割り当てる方式が多い
- メモリにアクセスするのと同じ命令で、I/Oをread、writeする

【発展】
記憶保護

- 記憶空間を保護したい
  - コード領域とデータ領域を別々にして、互いに保護
  - OSが扱うシステム領域とユーザタスクが扱うユーザ領域を別々にして、ユーザタスクがシステム領域を触れないように保護

- メモリ保護機能
- ユーザタスクを他のタスクから保護したい
  - 論理記憶空間の導入
  - MMU（メモリ管理ユニット）の利用

【発展】
論理記憶空間

- 物理メモリ領域を再配置して、物理アドレスと独立した論理的なアドレスをメモリに与えられる
- 再配置の単位
  - 固定長（ページ）➔ ページング（Paging）
  - 可変長（セグメント）➔ セグメンテーション（Segmentation）
- 論理空間を複数用意
  - 互いに保護したい単位で論理空間を分離
  - 論理空間単位で記憶保護を行う。

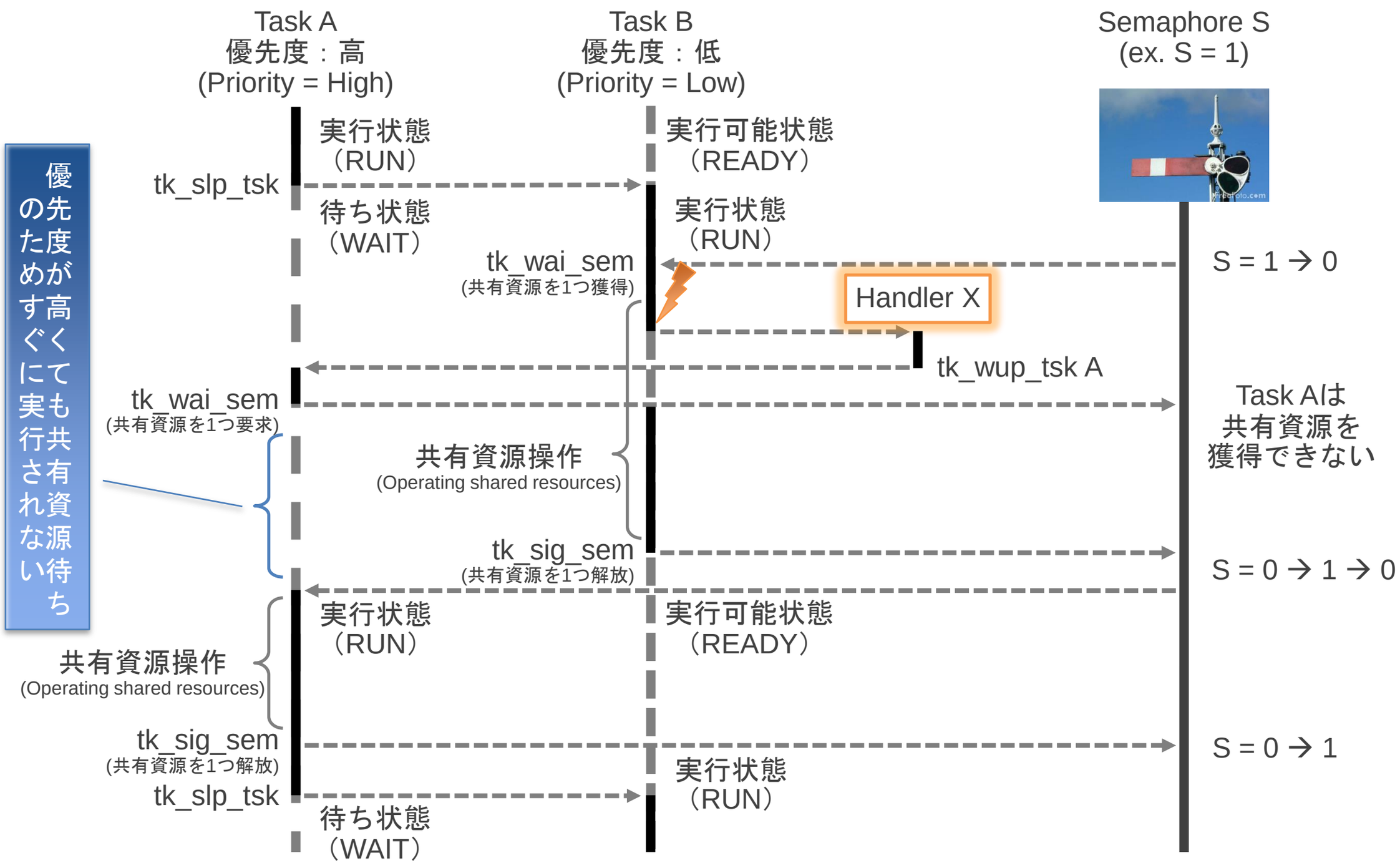
【発展】
仮想記憶

- メモリ量の制約を気にせずにプログラミング
  - もともと大型計算機での要求。でも便利は確か。
- 実メモリ量よりも大きな論理アドレス空間を提供する機能＝「仮想記憶」
- 実メモリに格納しきれない情報は二次記憶に退避（swapping）
  - 二次記憶装置が必要
  - swapping処理によってリアルタイム性が損なわれる

第５章（発展、参考）
リアルタイムシステムの高度な課題

1　優先度逆転
Priority Inversion

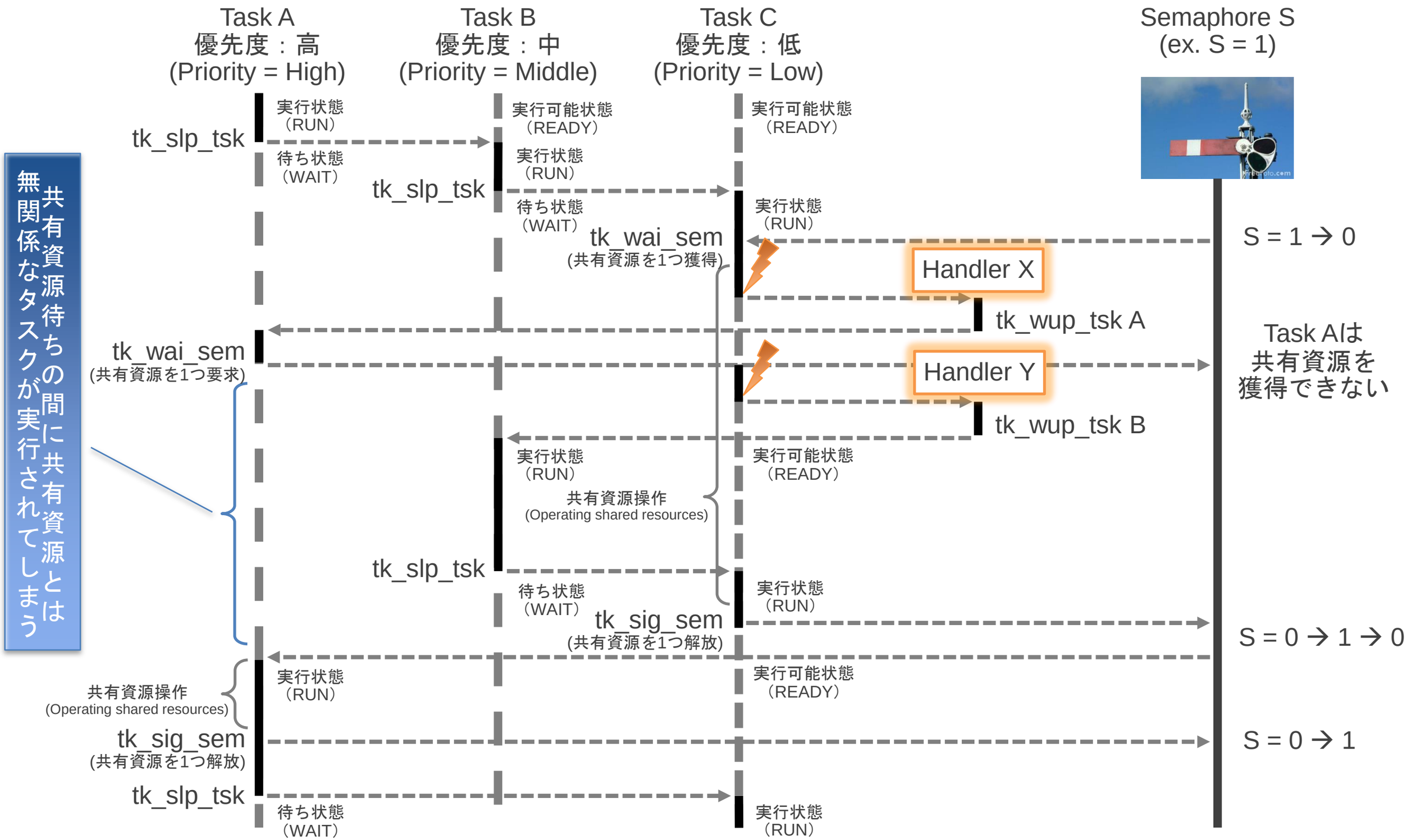
排他制御による高優先度タスクの待ち



優先度逆転(Priority Inversion)

- 排他制御のため低優先度タスクが高優先度タスクをブロックしている間に、排他制御とは無関係な中優先度タスクが低優先度タスクに優先して動作してしまい、その結果、高優先度タスクがあたかも低優先度タスクであるかのように後回しにされてしまう現象。
- 通常は正常に動作しているが、特定のタイミングのときだけ、設計者の意図に反して高優先度タスクの処理が予想外の時間ブロックされ、結果として制限時間内に処理が完了できなくなってしまうことがある。
- 発生条件
  - 高優先度タスク、中優先度タスク、低優先度タスクがある。
  - 高優先度タスクと中優先度タスクは割り込みハンドラによって起床される。
  - 高優先度タスクと低優先度タスクは資源を共有しているため 排他制御を行っている。

優先度逆転による高優先度タスクの遅延



優先度逆転の回避策

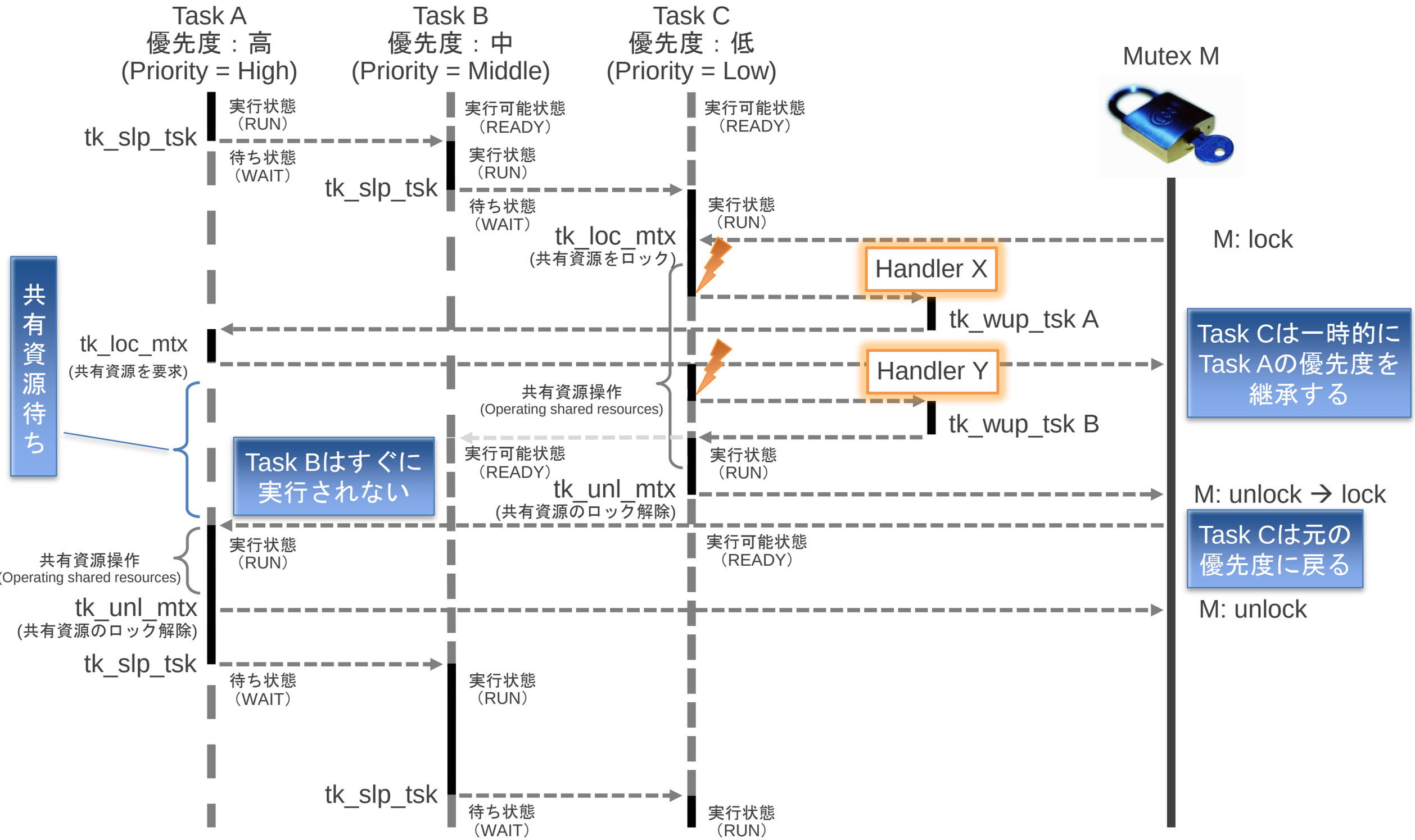
- 排他制御が不要となるタスク構成を検討する。
- ミューテックスの優先度継承プロトコル、または優先度上限プロトコルを使用する。
- 永久待ちにせず、タイムアウトを指定する。

ミューテックス(Mutex)

- 機能
  - 共有資源を使用する際にタスク間で排他制御を行うためのオブジェクト
  - 排他制御に伴う上限のない優先度逆転を防ぐための機構として、優先度継承プロトコル(priority inheritance protocol)と優先度上限プロトコル(priority ceiling protocol)をサポート
- 優先度継承プロトコル(priority inheritance protocol)
  - 低優先度タスクが高優先度タスクをブロックしたとき、低優先度タスクが一時的に高優先度タスクの優先度を継承して中優先度タスクに割り込まれないようにすること。つまり、低優先度タスクは排他制御処理中、高優先度タスクをブロックしたときだけ高優先度タスクの優先度を継承して優先的に動作し、排他制御処理が終了したら元の優先度に戻る。

- 優先度上限プロトコル(priority ceiling protocol)
  - 資源を共有するタスク群の最高優先度を排他制御処理を実行する際の優先度とし(上限優先度(ceiling priority)の設定)、排他制御処理中は資源を横取りできないようにすること。つまり、排他制御処理を開始したタスクは、それまでの優先度にかかわらず常に上限優先度に変更されて優先的に動作し、排他制御処理が終了したら元の優先度に戻る。
  - 優先度上限プロトコルは優先度継承プロトコルがベースとなっており、優先度継承プロトコルでは低優先度タスクが高優先度タスクをブロックしたときのみ優先度が変更されるのに対し、優先度上限プロトコルでは排他制御処理を開始すると常に優先度が変更される。

ミューテックスによる優先度逆転の回避

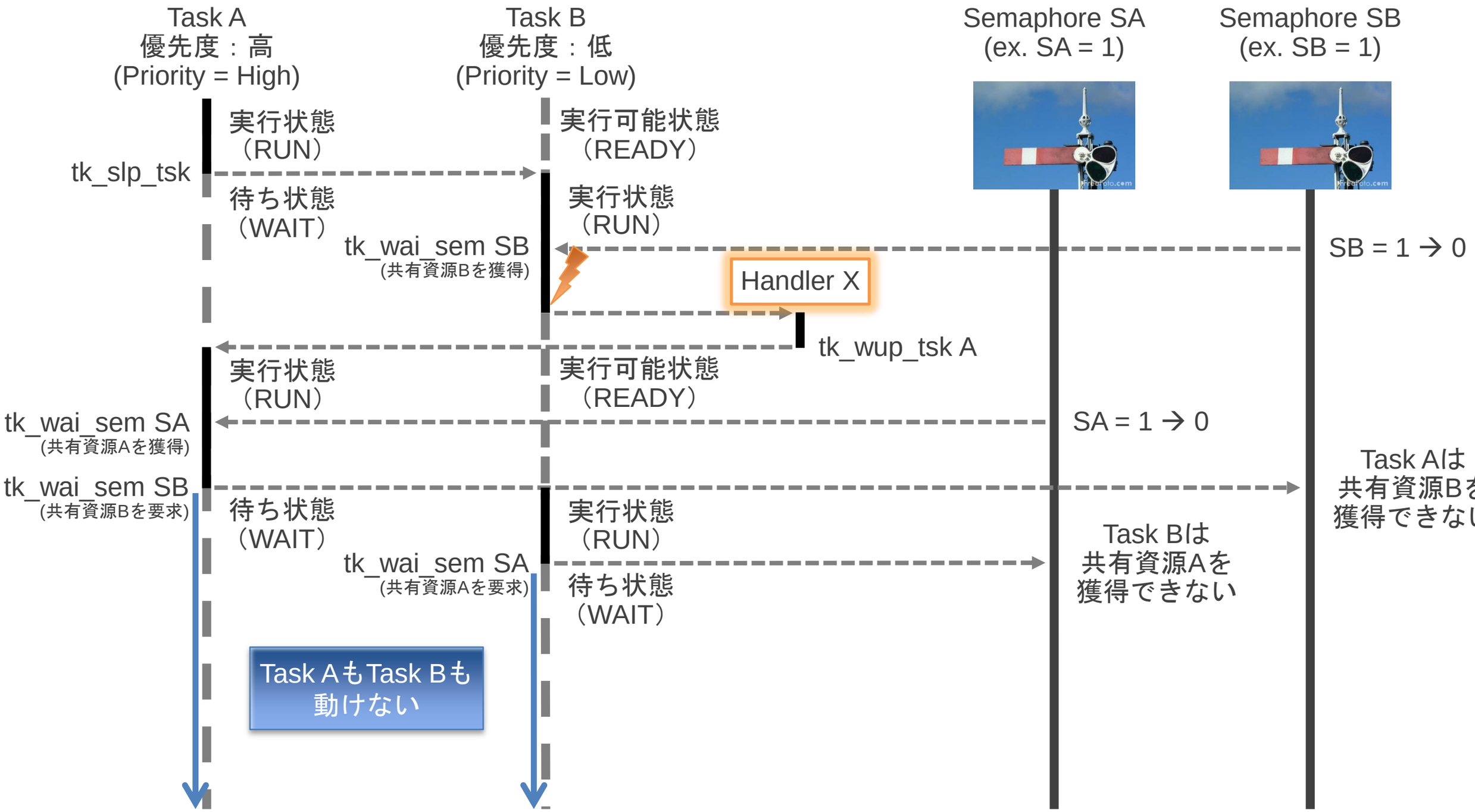


2　デッドロック
Deadlock

デッドロック(Dead Lock)

- 2つ以上の資源を共有するタスク群が互いに互いの実行をブロックしてしまい、永久に動作できなくなってしまう現象。
- 通常は正常に動作しているが、特定のタイミングのときだけ、設計者の意図に反して資源を共有しているタスク群が動作できなくなり、システムの一部の機能が停止してしまう。
- 発生条件
  - 2つまたはそれ以上の資源を優先度が異なるタスク間で共有し、排他制御を行っている（セマフォを使っているとは限らない）。
  - 高優先度タスクは割り込みハンドラによって起床される。
  - 共有資源を同時に（ネスティングして）要求しており、その要求順序が一定でない。

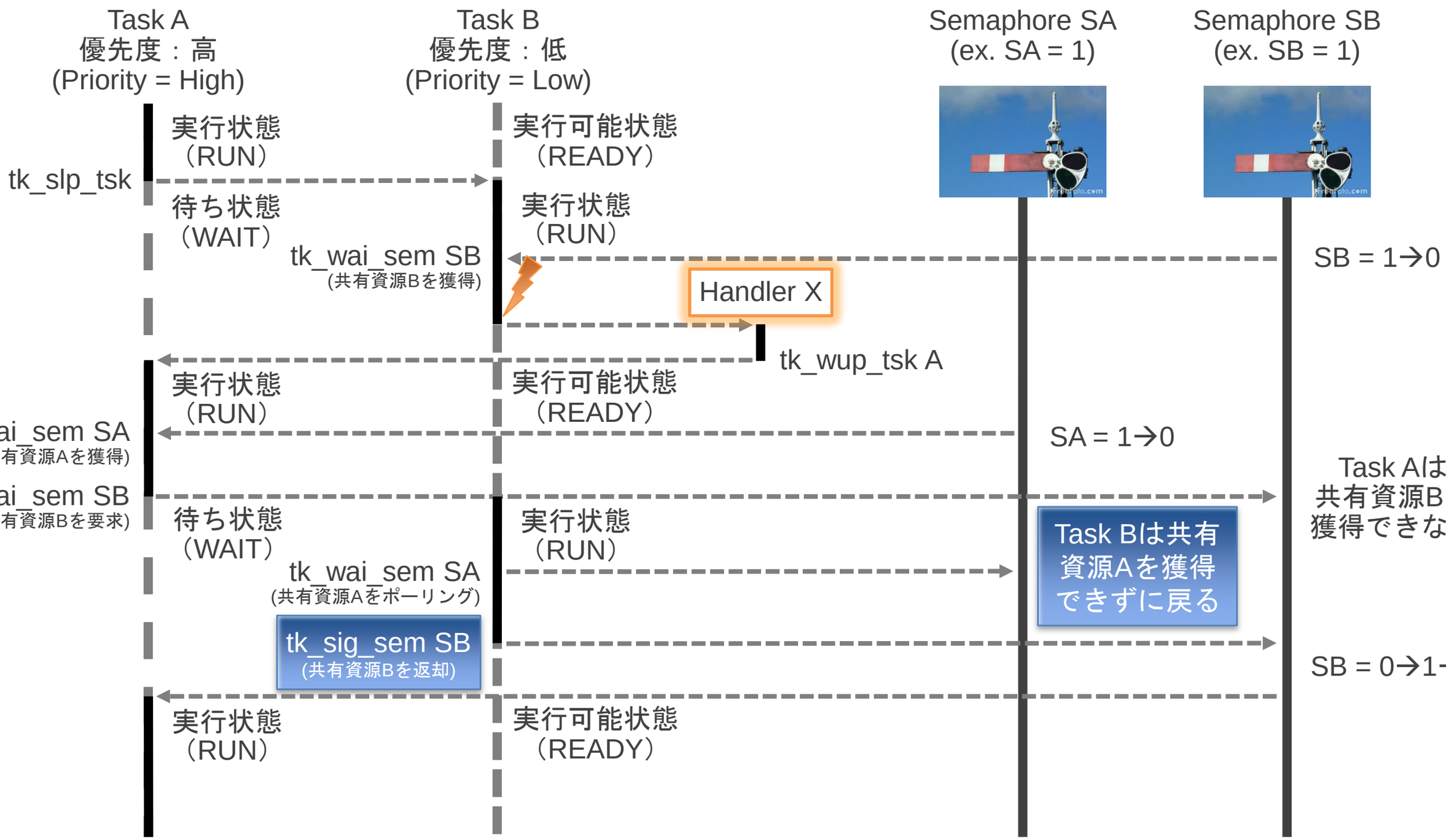
デッドロックによるタスクの停止



デッドロックの回避策

- 共有資源の獲得順序を一定にする。
- 関連する資源は1つにまとめて管理する。
- 永久待ちにせず、ポーリングを指定する。
- ミューテックスの優先度上限プロトコルを使用する。

ポーリングによるデッドロックの回避



第６章
なぜリアルタイムOSか？

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。
- そのための技術
  - 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

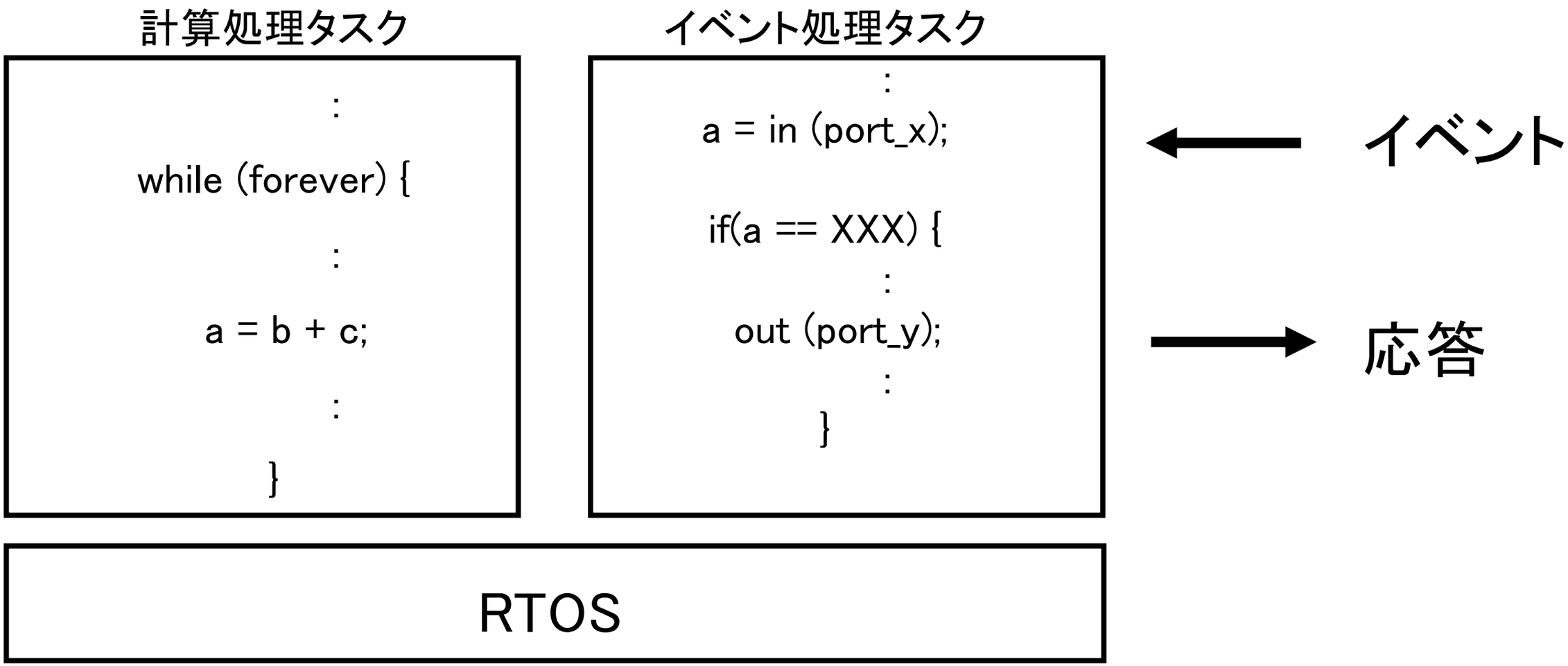
なぜRTOSが必要か？

- これを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。

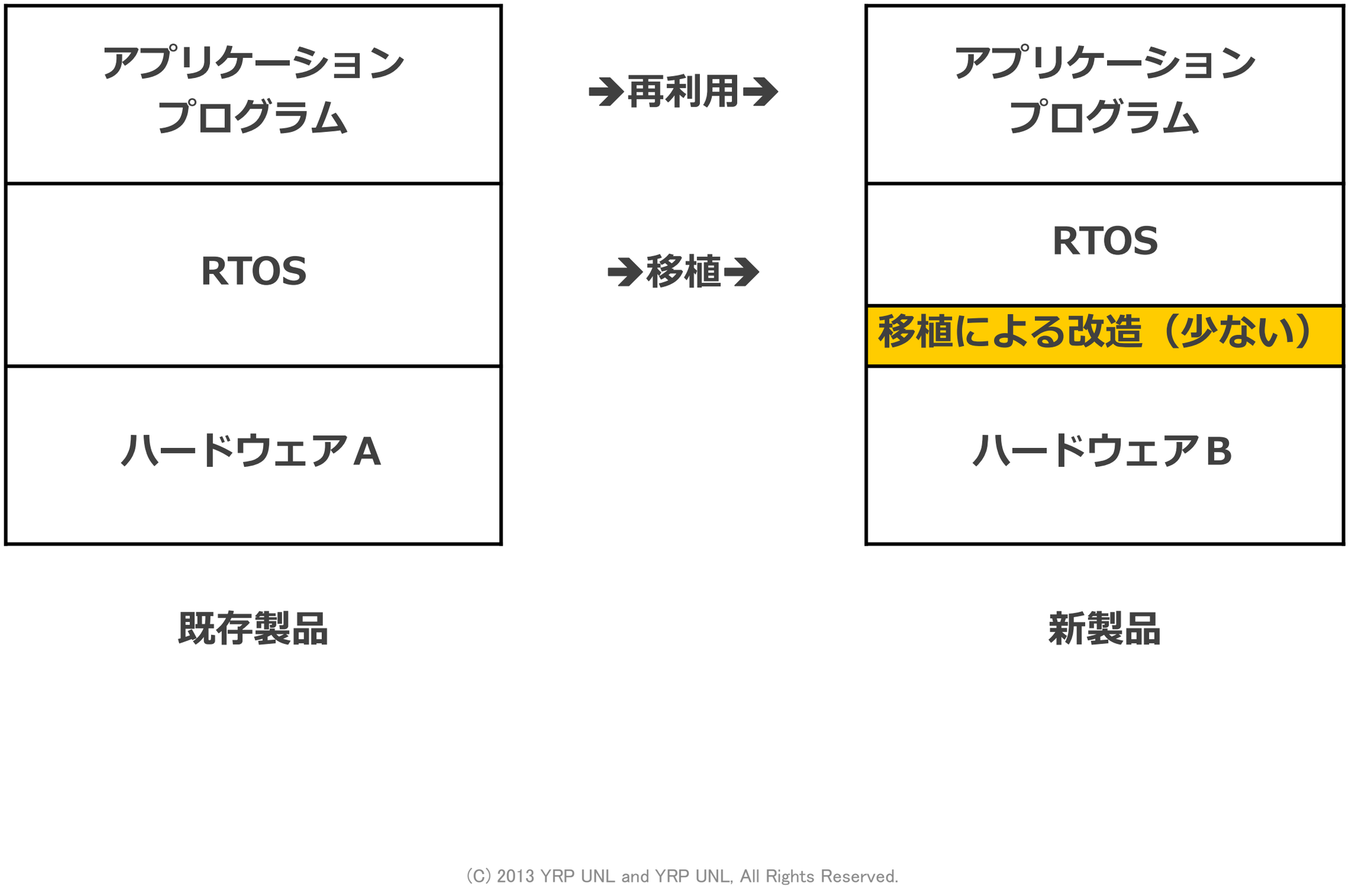
- 何らかの汎用的なシステムでサポート
  ➔ リアルタイムOS（Real-time OS: RTOS）

RTOS導入のメリット
１．プログラム作成が容易に

- 機能毎にタスク分割して開発



RTOS導入のメリット
２．プログラムの再利用性



RTOS導入のメリット
３．保守性・拡張性の向上

- システム全体のモジュール性が向上
  - 他への影響を抑えて拡張できる
  - モジュール単位で保守できる
  - 責任分界点も明確に

- 関連製品の利用
  - ミドルウェアや各種ソフトウェア資産の利用
  - 開発支援ツールの利用

第７章
まとめ（RTOS）

RTOSとは何か？

- リアルタイム・組込みシステム開発において、
  共通に使用される管理プログラム
- リアルタイムシステム向きの機能を持つ
  （各イベントに対して高速に応答できる）
- タスク切換時間、各サービスコール時間があらかじめ予測できる
- コンピュータの持つ資源を仮想化し、効率利用できる（再利用性）

リアルタイムシステムの必要な機能とRTOS

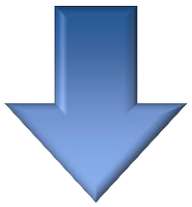
RTOSが提供する機能

- タスク管理機能
- タスク同期管理機能
- 同期通信機能
- メモリ管理機能
- 時間管理機能
- 割込み管理機能

RTOSが提供しない機能

- 入出力管理機能
- 通信・ネットワーク機能
- ファイル管理機能
- ユーザインターフェース機能（GUI)
- 音声認識、音声合成
- 画像圧縮伸張

など



- 上位のミドルウェアでサポート

【実習】μT-Kernel入門(協力:スパンション・イノベイツ)テキスト
「リアルタイムOS概要とμT-Kernelの基本機能解説」

著者　　T-Engine 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@t-engine.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。