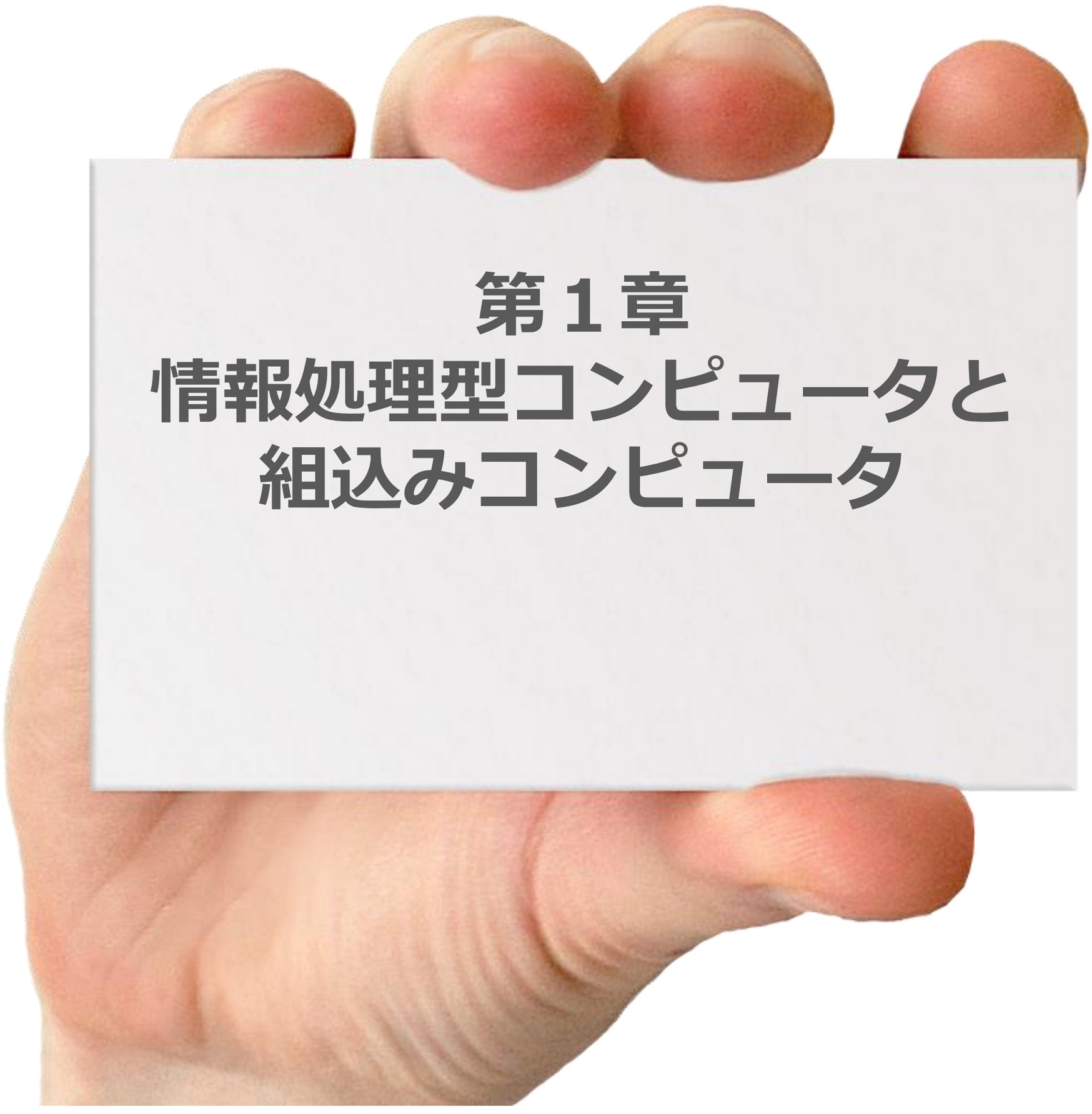
組込みリアルタイムOS入門

2016年度

YRP Ubiquitous Networking Lab.
TRON 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

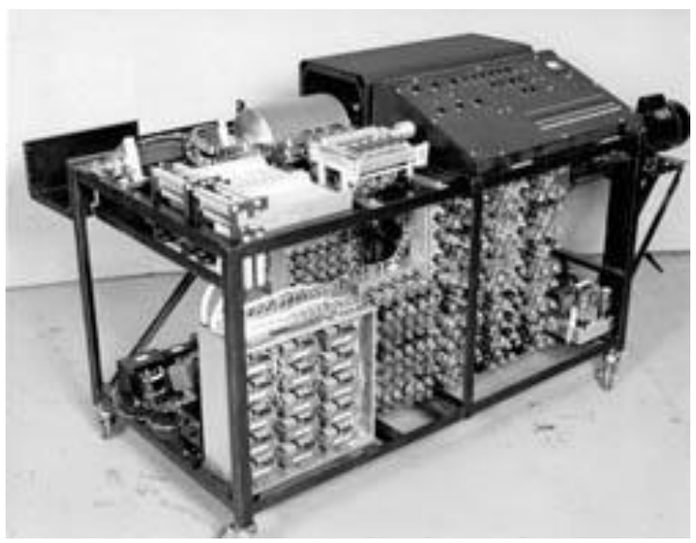
情報処理型のコンピュータ（１）

- パーソナルコンピュータ
- サーバ
- スーパーコンピュータ

- コンピュータらしいコンピュータ

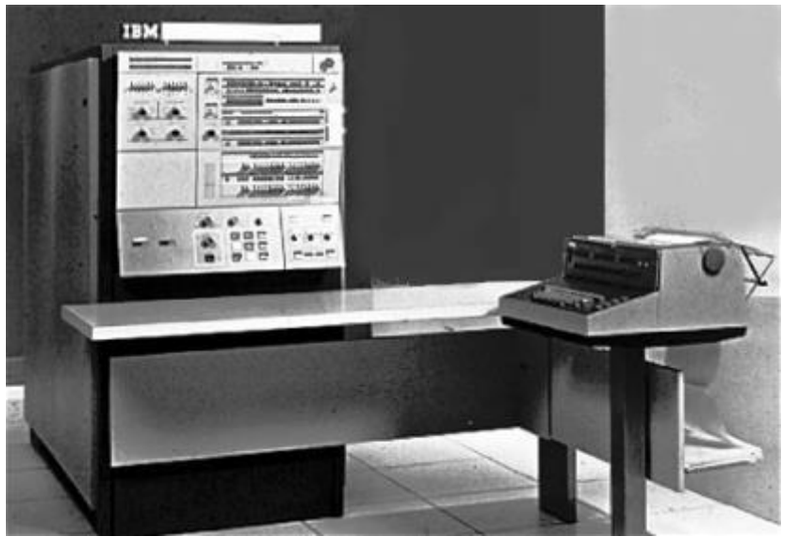
情報処理型のコンピュータ（２）

ENIAC



Atanasoff-Berry Computer

“K”（京）スーパーコンピュータ



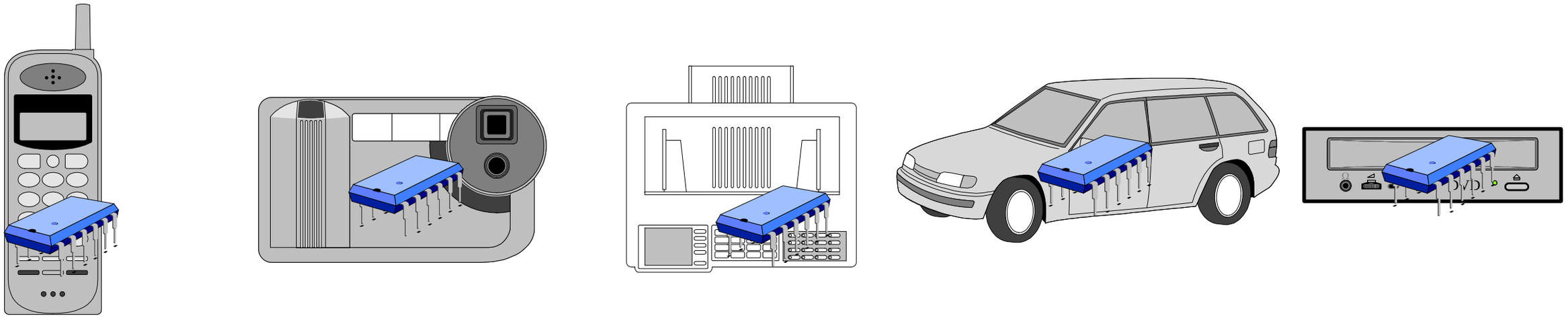
IBM 360

Cray 1

Apple II

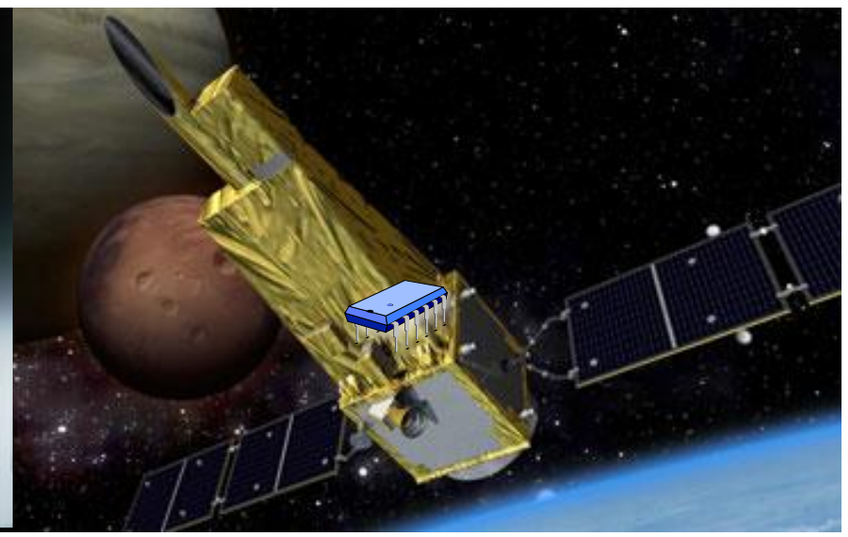
組込み型のコンピュータ（１）

- 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）

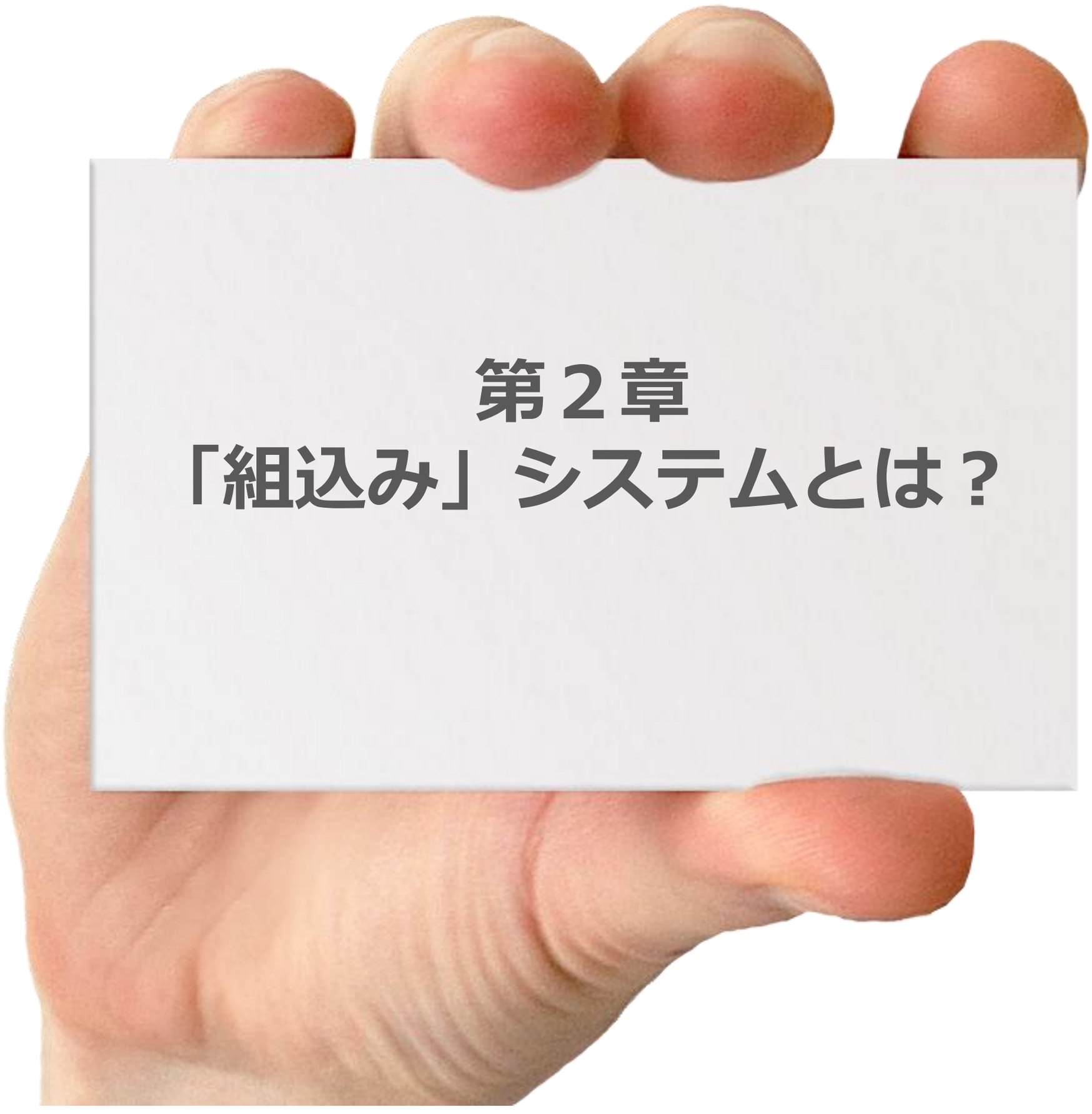
組込み型のコンピュータ（2）



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。



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、通信処理が中心
  - 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**
  - 製品ロット数が大きくなると、ハードウェアコストの割合が増大
  - 価格競争に勝つには、（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

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。

第３章
「リアルタイム」システム
Real-time System

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）**

リアルタイムシステムとは？（１）

- 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「素早い」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

- どこまで早ければ、「素早い」のか？

リアルタイムシステムとは？（２）

- システムに与えられた時間制約条件を満たすために十分な素早さをもっていればよい。
- 重要なポイントは「時間制約条件を満たす」こと
- 高速なCPUだけでは、時間制約条件を満たすことは困難。
  - ソフトウェアやプログラム上で、時間制約を扱うことができる必要がある。
  - それがリアルタイムシステム

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

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

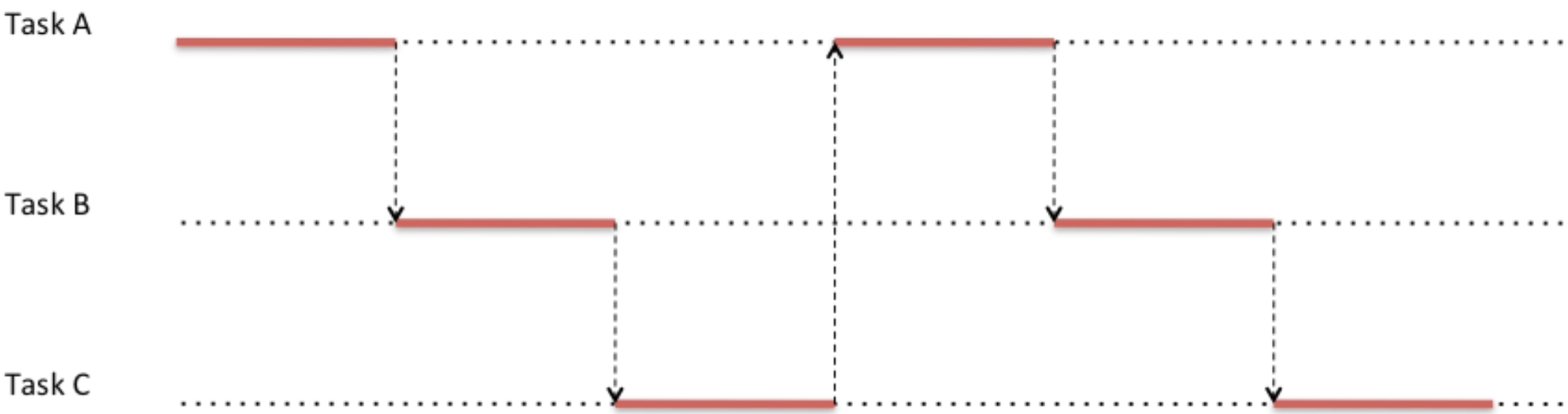
スケジューリング（scheduling）

- 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。



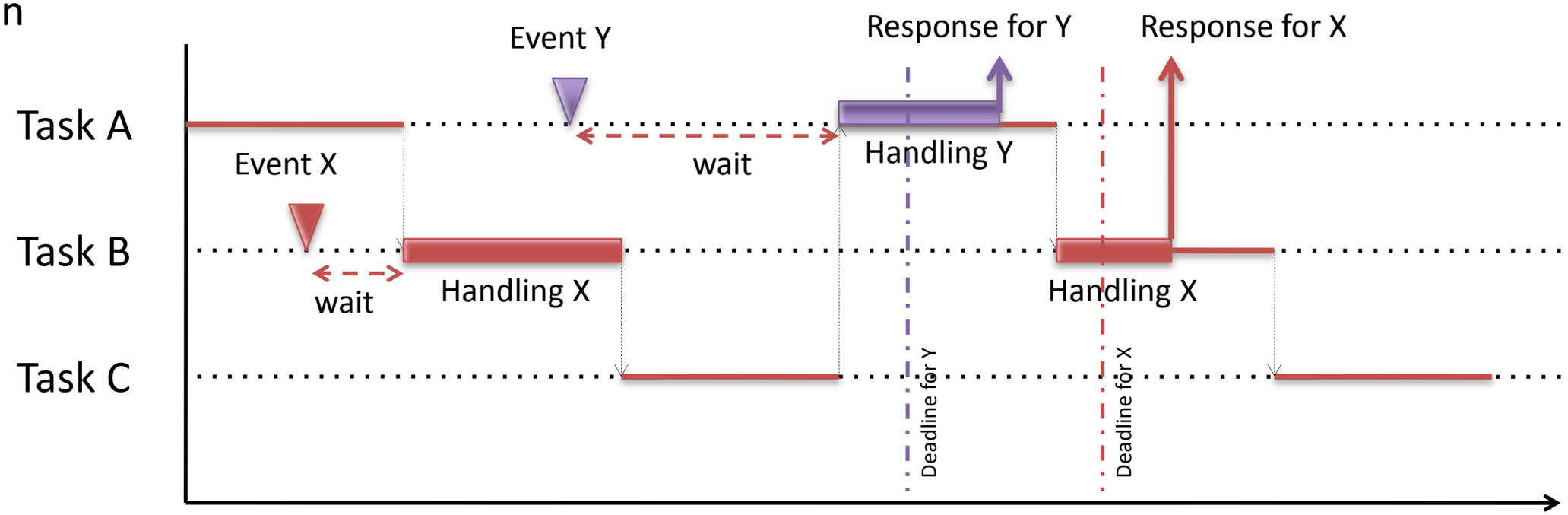
デッドラインが先の処理を先に行なうスケジューリング

- 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）

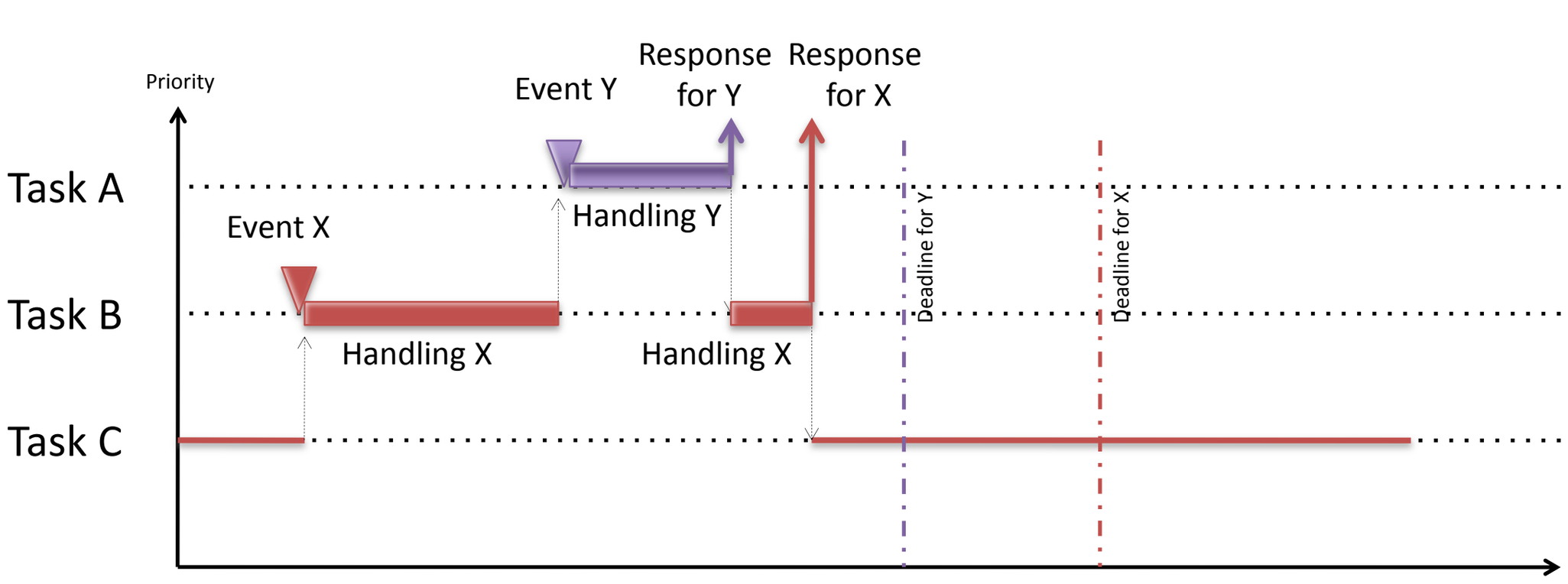
- 優先度の高い処理は、優先度の低い処理より優先して実行されるため、先に終えることができる

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

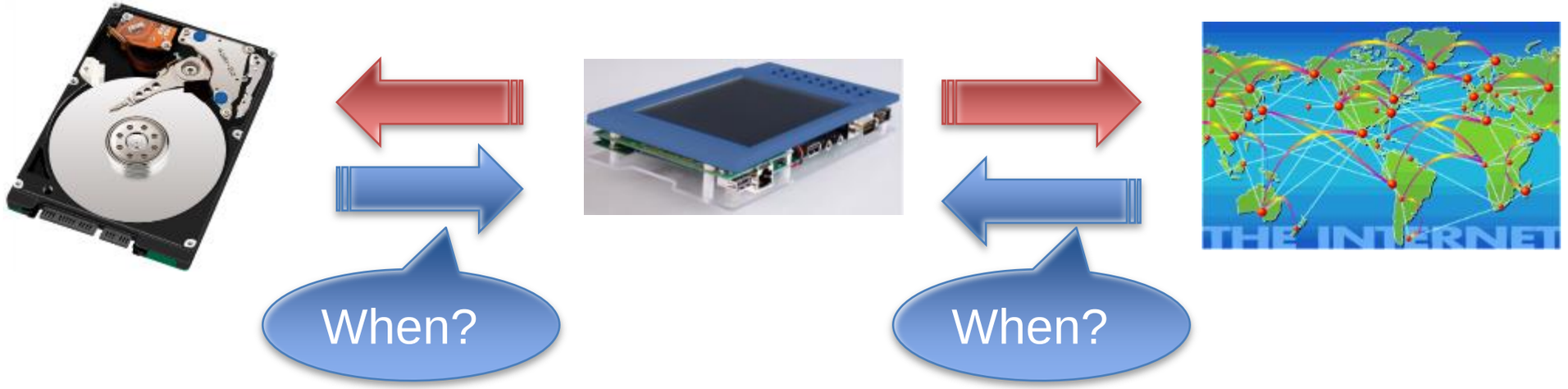
処理時間を予測できるためには？

- 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

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

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
- **処理の予測可能性に基づいたスケジューリング**
  - タイムアウト機能による最悪時間保証

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

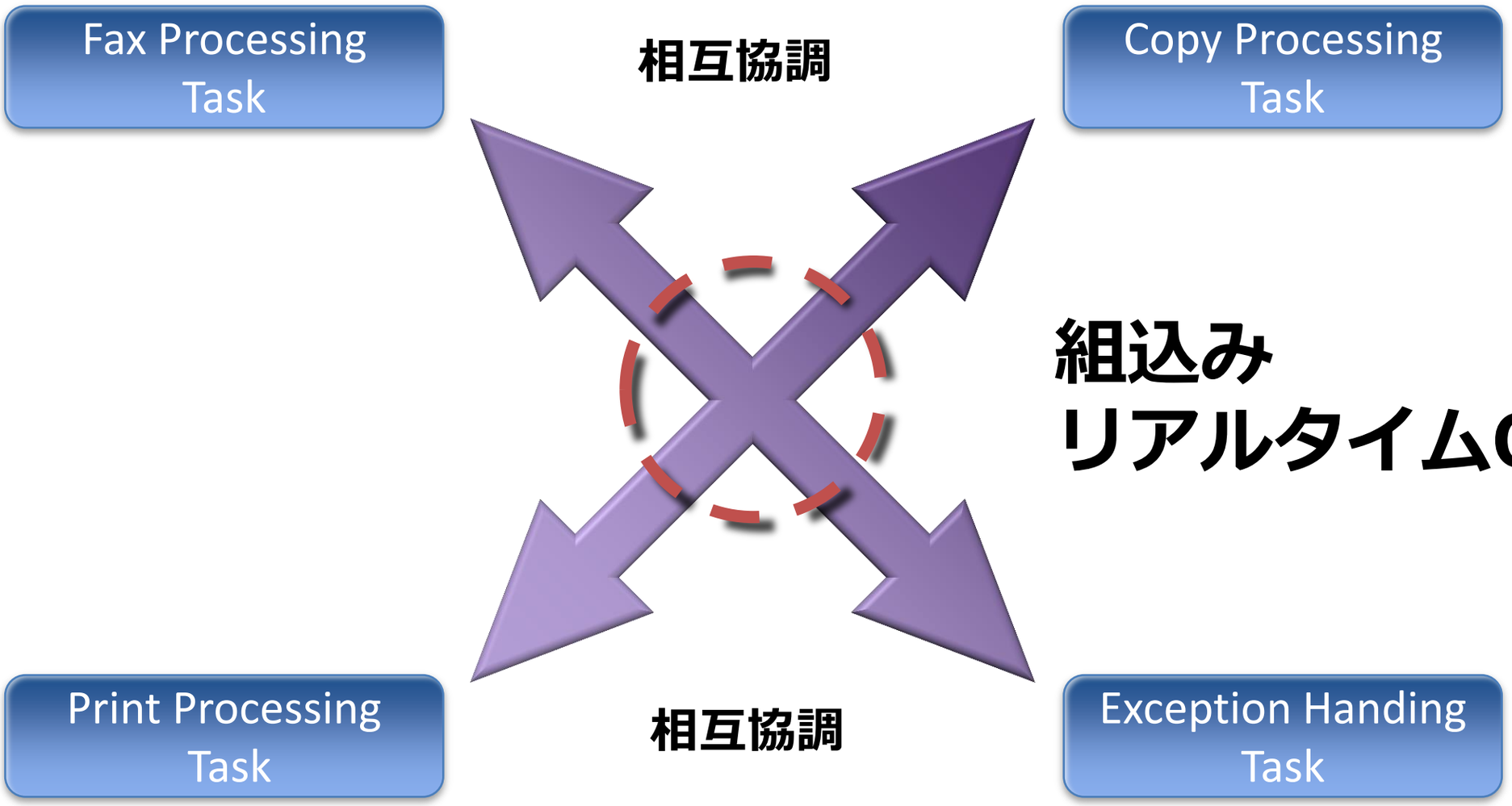
Exception Handing Task

[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

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

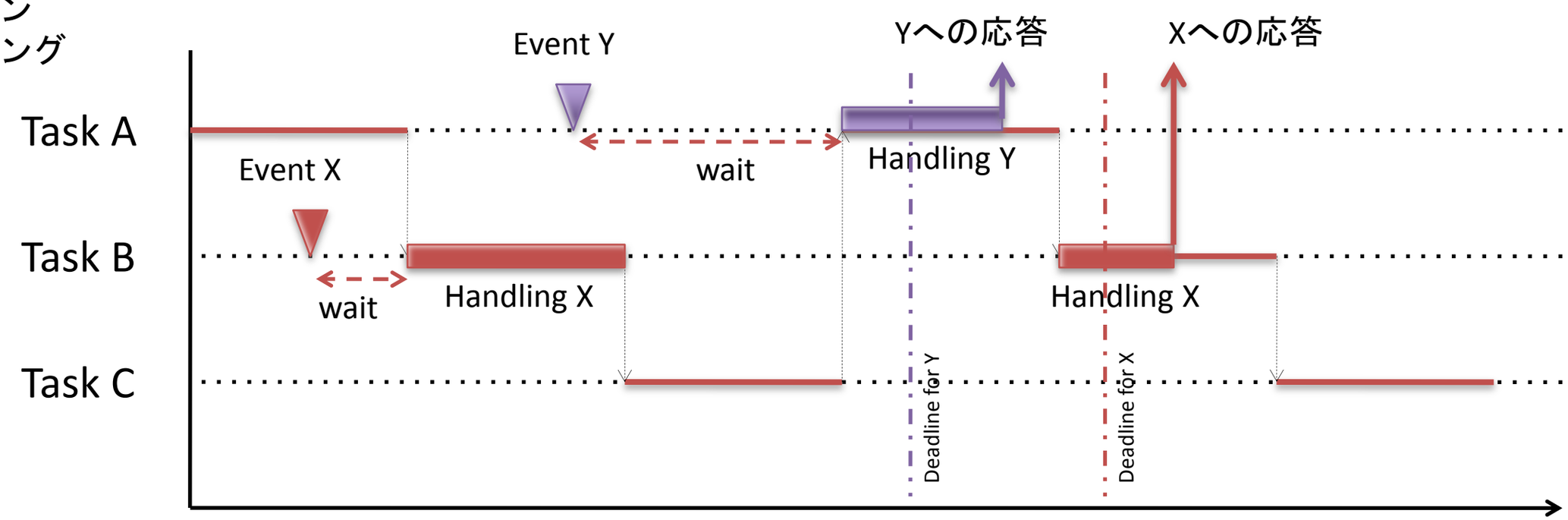
リアルタイム方式（Real-time Scheduling）

- 以下の方式を融合したものが一般的

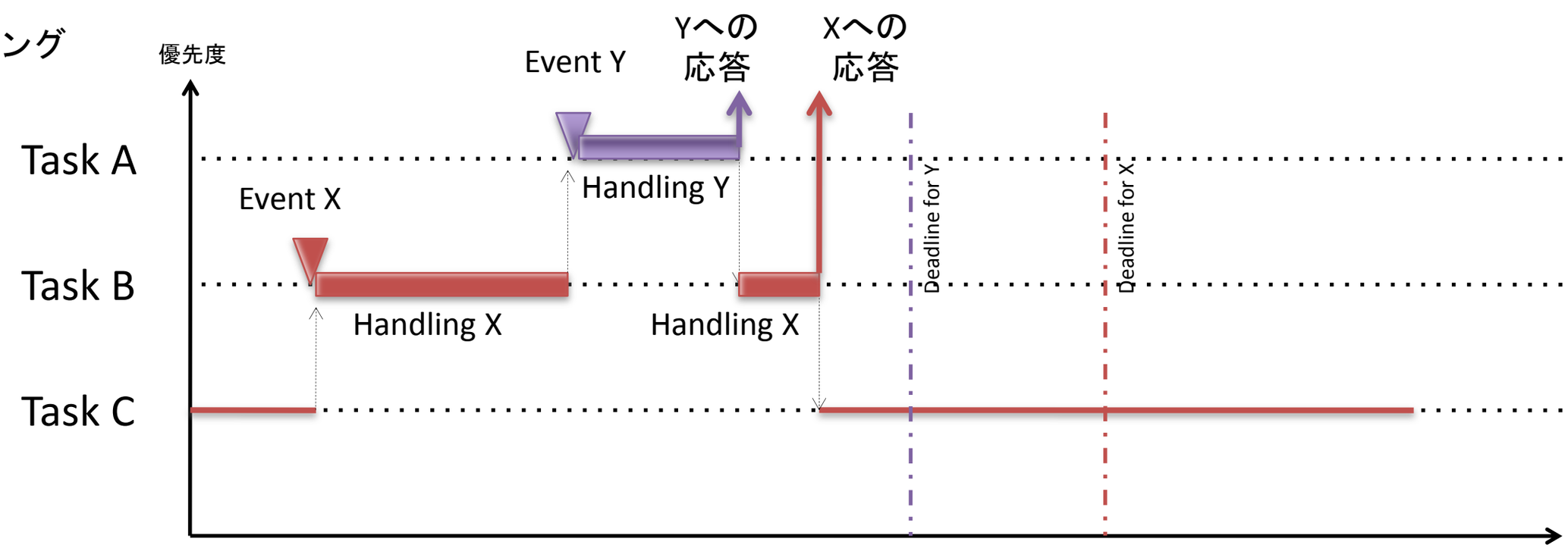
- 優先度ベース（Priority-Based Scheduling）
  - A task can have a priority.
  - Tasks with higher priorities can be allocated a CPU in prior to tasks with lower priorities.
- 横取り型（Preemptive Scheduling）
  - A scheduling is called preemptive if it is capable of forcibly removing processes from a CPU when it decides to allocate that CPU to another process.
- イベント駆動型（Event-Driven Scheduling）
  - Event-driven scheduling switches tasks only when an event of higher priority needs service.
    - This is also called preemptive priority, or priority scheduling.

リアルタイム方式とラウンドロビン方式

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

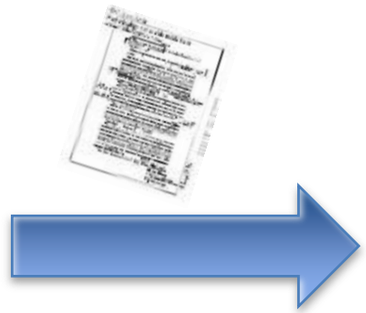
- **タスクの間の相互関係に合わせて、処理の実行を調節すること → 同期（Synchronization）**

【相互関係１】仕事の依存関係による同期

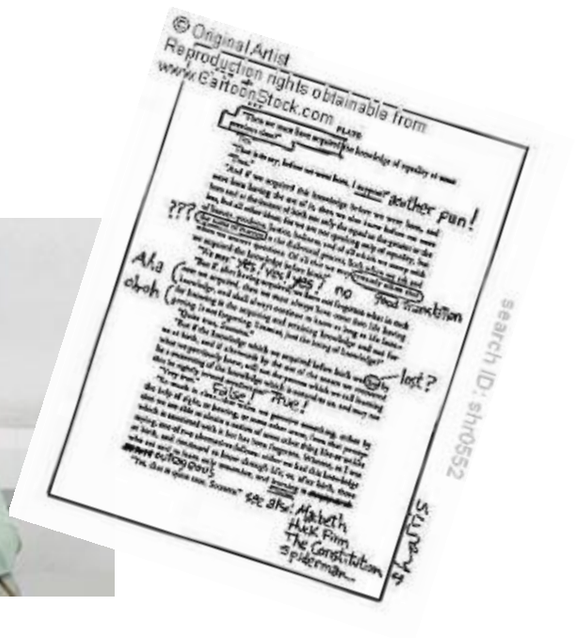
- **依存関係**
  - Aの仕事がおわらないと、Bの仕事が始まらない。
    （例）学生のレポートが提出されて、初めて教師は採点する。

- **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



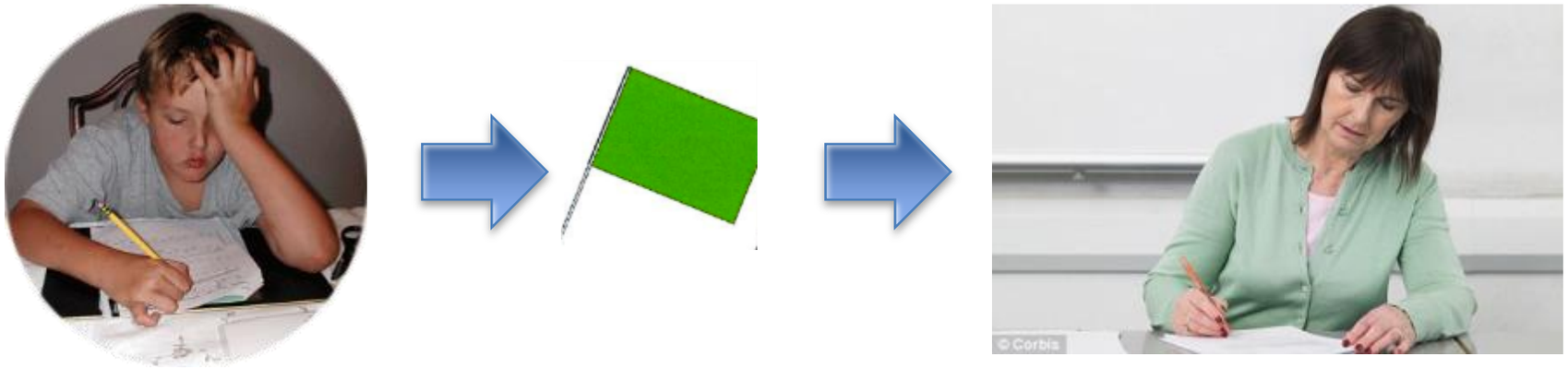
宿題



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

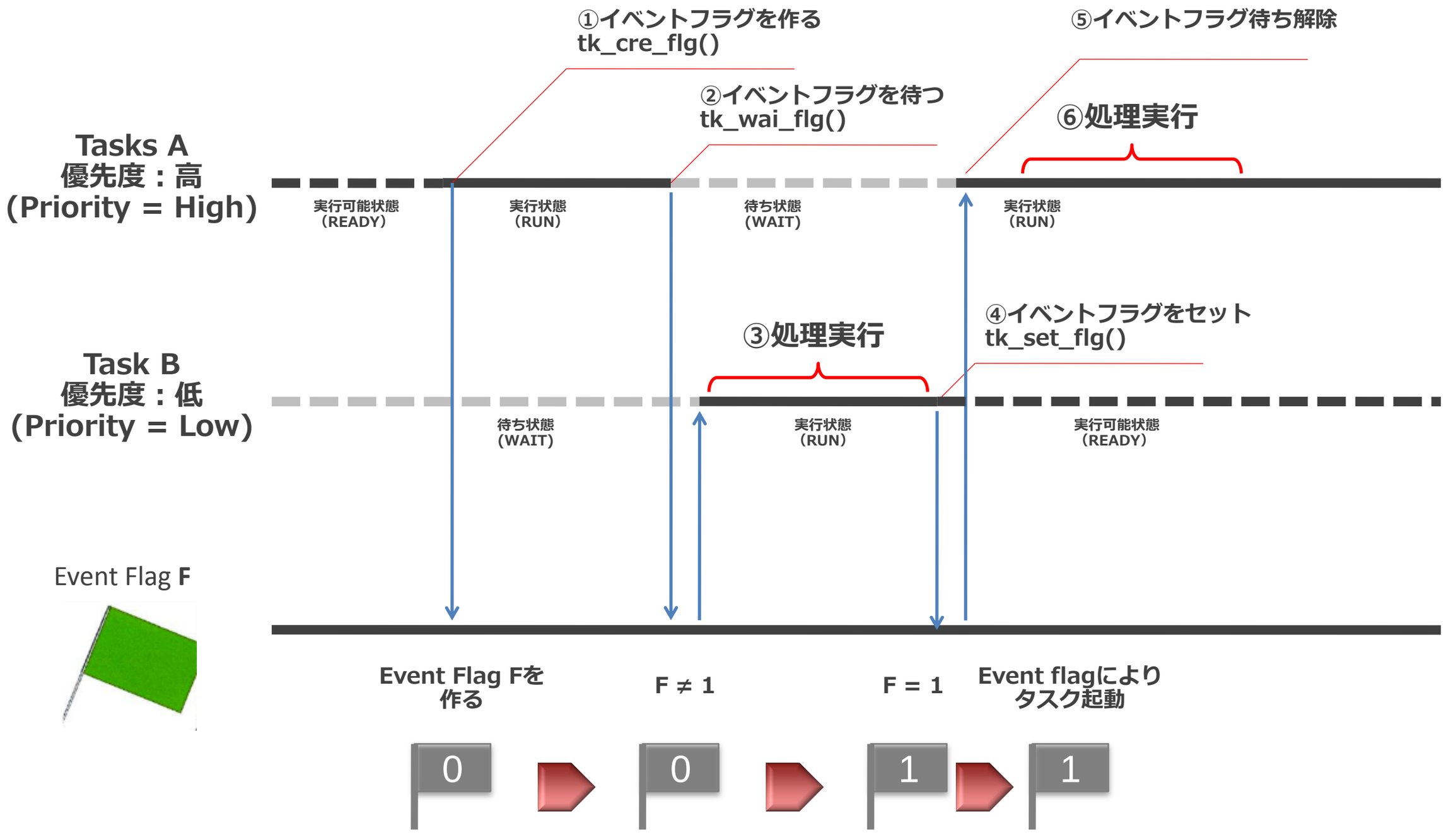


- **イベントフラグの機能**
  - 事象の通知をタスク間で通信する
  - ビットパターンを事象に割り当てることで利用
  - OR待ちやAND待ちなどの事象待ちが可能
  - パターンが一致すれば同時に複数のタスクを待ち状態から復帰可能

イベントフラグの基本機能

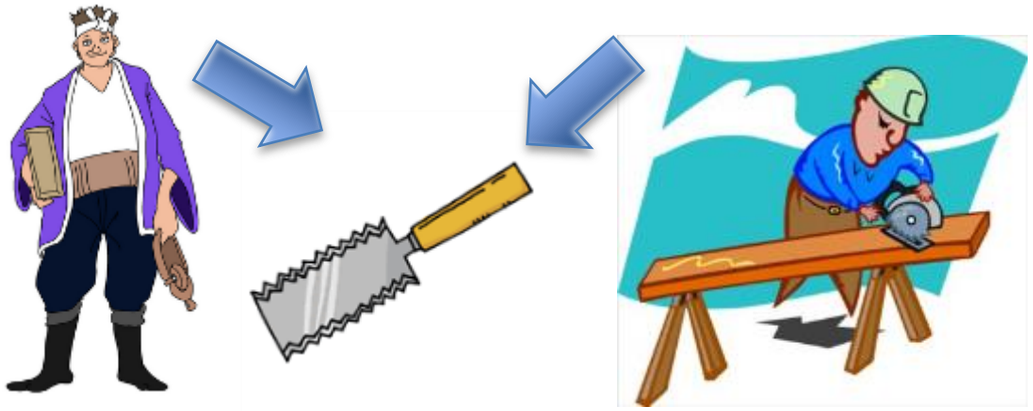
- **イベントフラグは、２種類の状態を持つ**
  - Set状態
  - Clear状態
- **イベントフラグへの基本操作**
  - セット（Set)命令 → イベントフラグは”Set”状態になる
  - クリア（Clear)命令→ イベントフラグは “Clear”状態になる
  - ウェイト（Wait)命令
    - イベントフラグが”Clear”状態
      → そのタスクをイベントフラグが”Set”状態になるまで待たせる。
    - イベントフラグが”Set”状態
      → そのタスクは動作をそのまま継続する
- **１つのイベントフラグは１ビットで実現可能**
  - 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

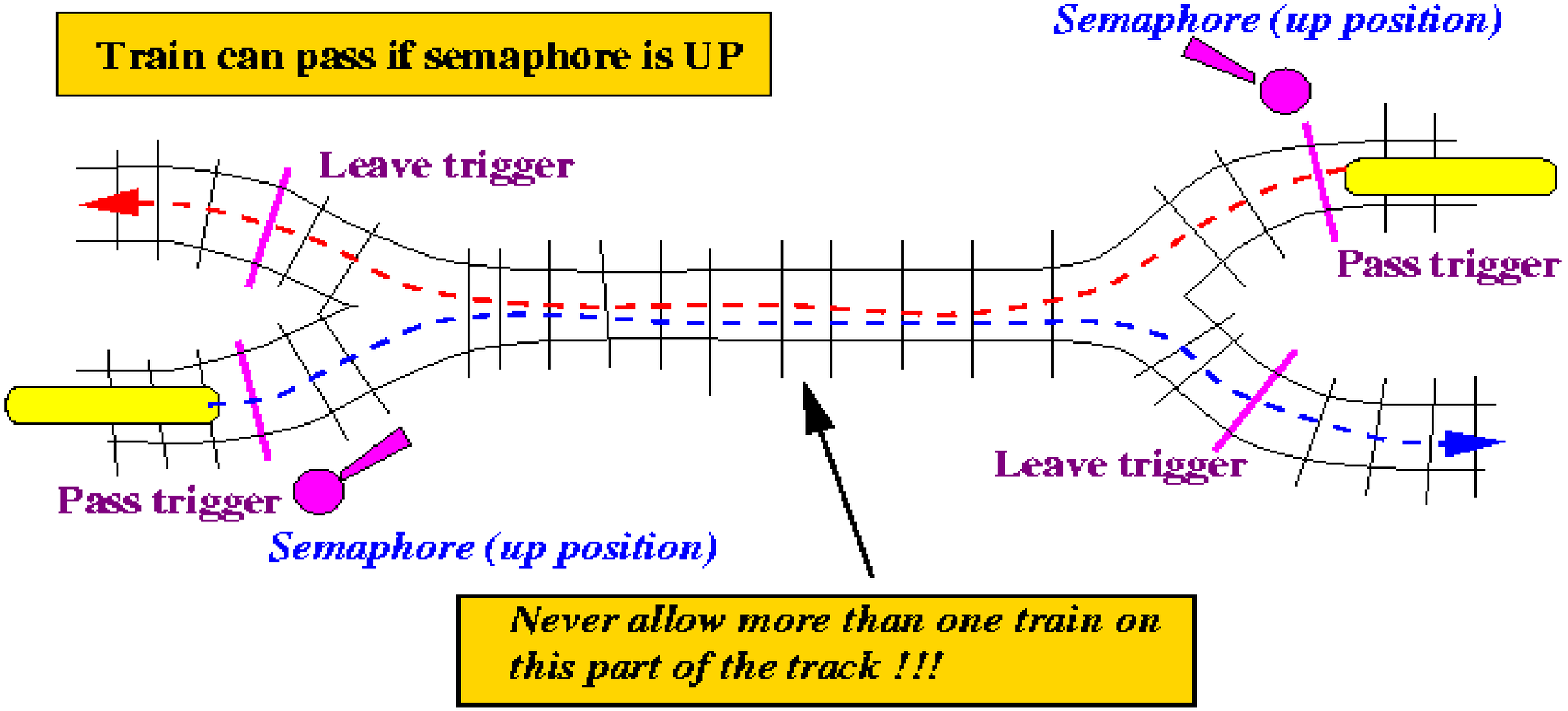


【相互関係２】道具を共有している時の同期 排他制御（mutual exclusion）

- **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（Semaphore）（1）



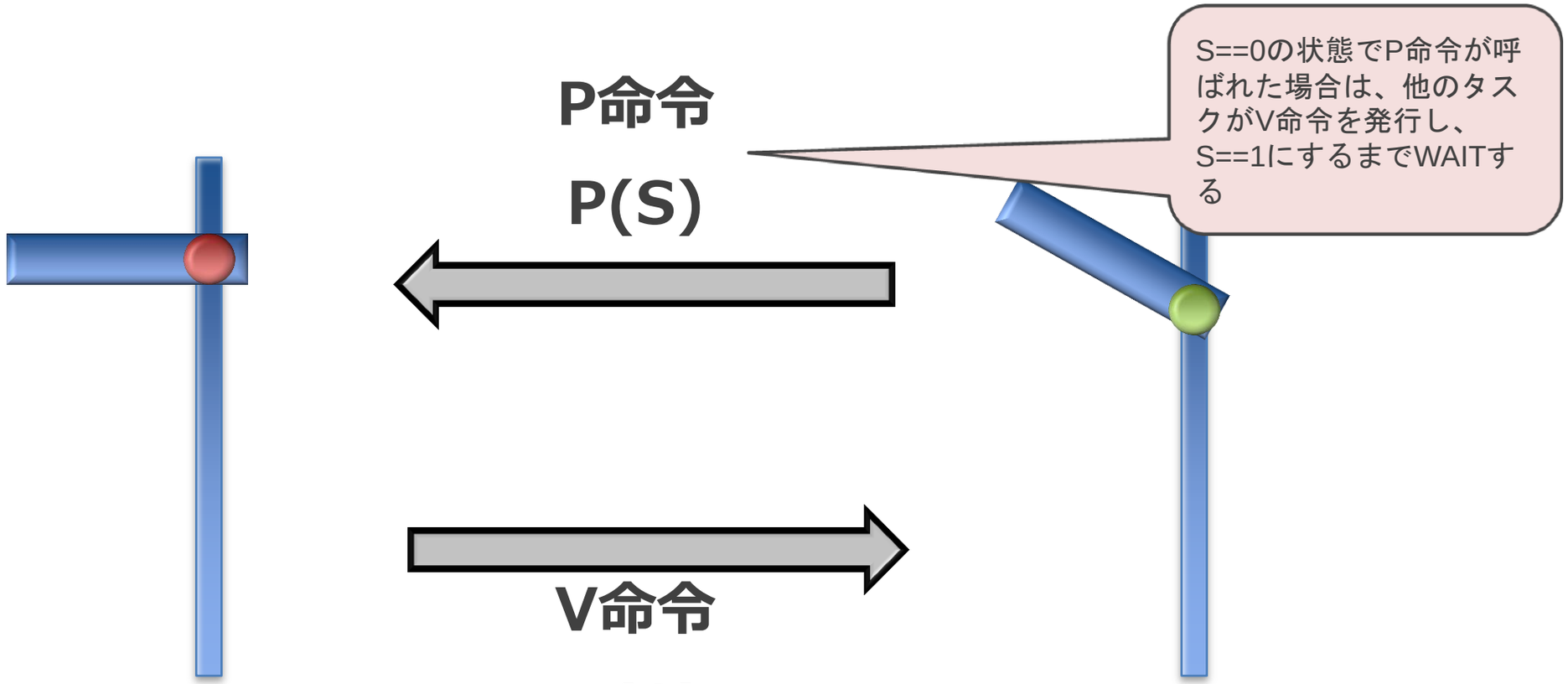
セマフォ（Semaphore）（2）

- 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

セマフォの基本動作（バイナリセマフォ）



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

- **バイナリセマフォ**
  - 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**
  - 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

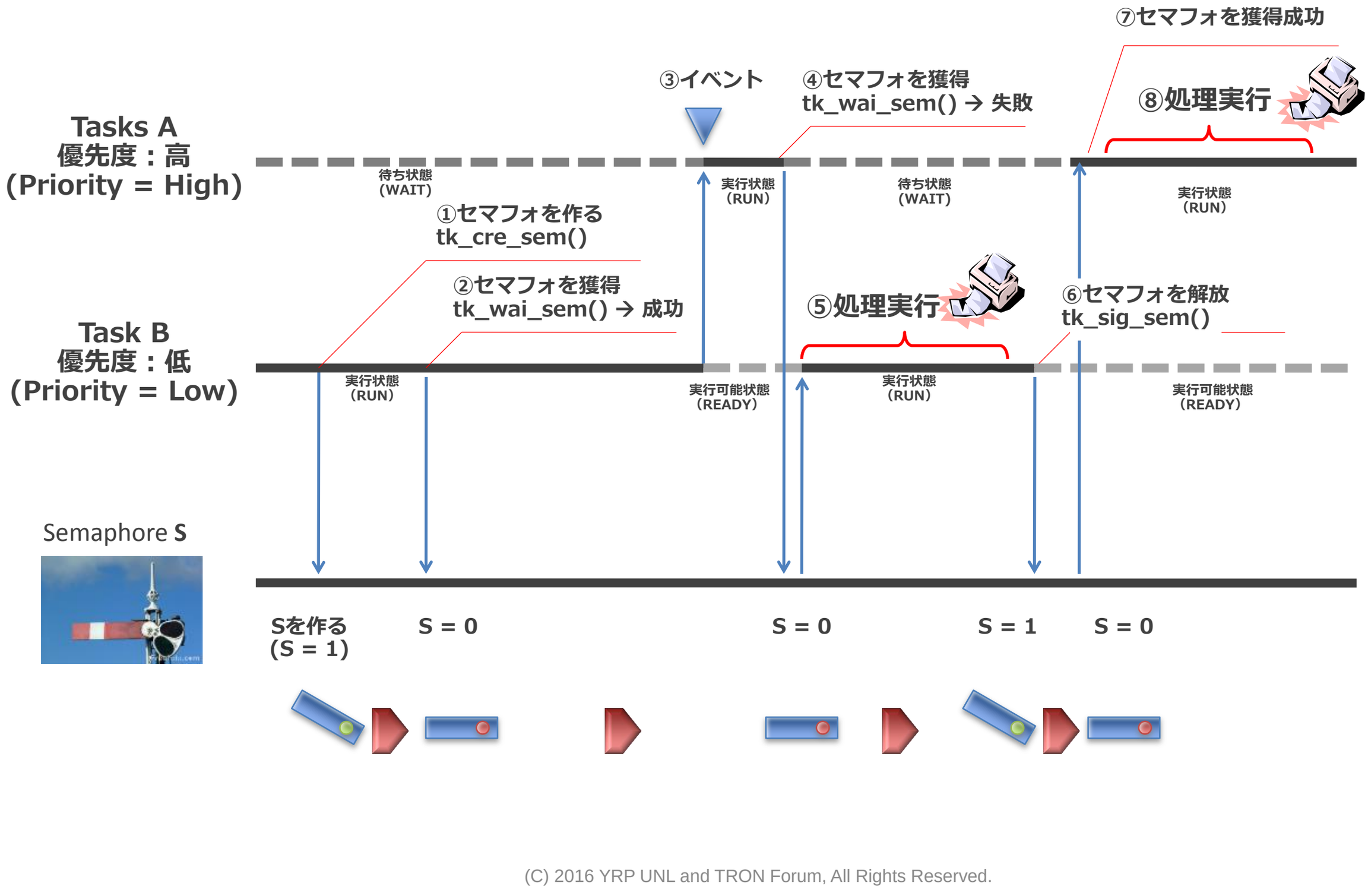
Task 1

```
:
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)
:
```

セマフォの例：Task B獲得→Task A獲得



3　通信

Communication

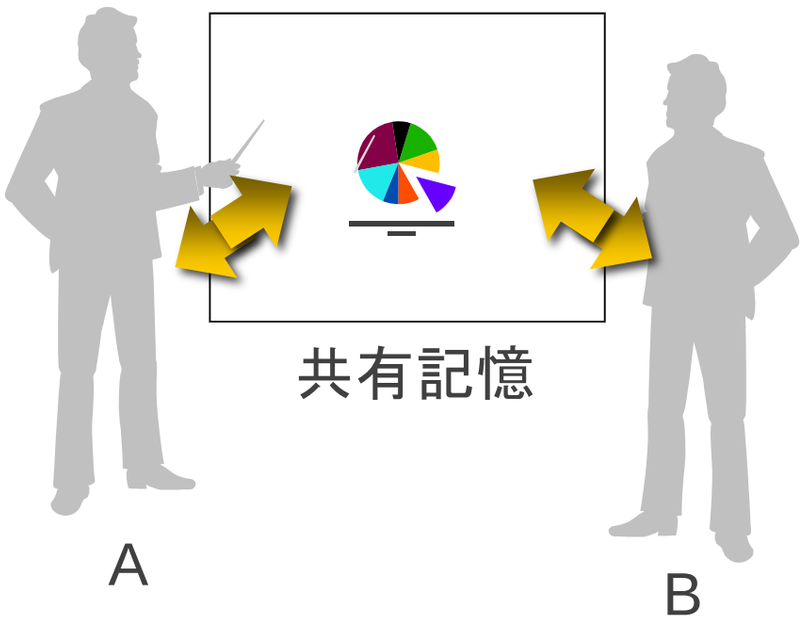
通信　（Communication）

- 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）

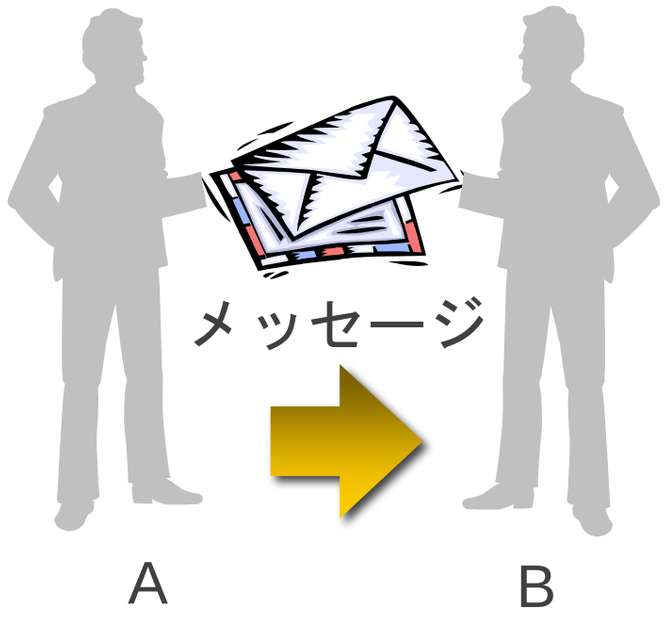
- 例：学生がレポートを書く➔ 教師はレポート採点
  - 学生･教師間の通信 ＝ レポートの内容

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



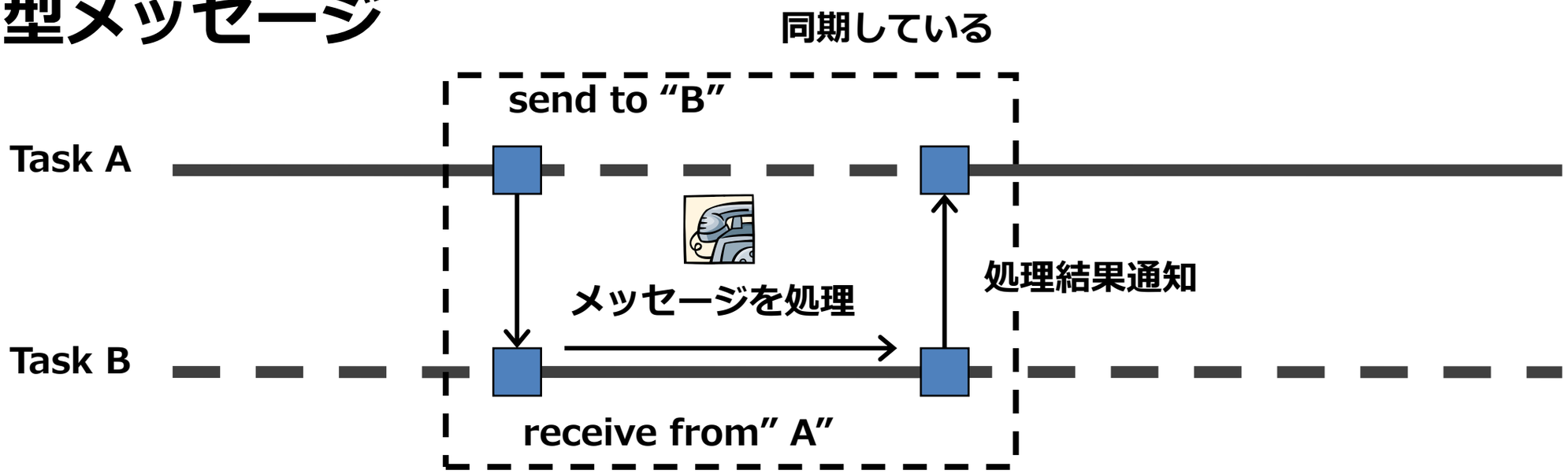
共有記憶方式



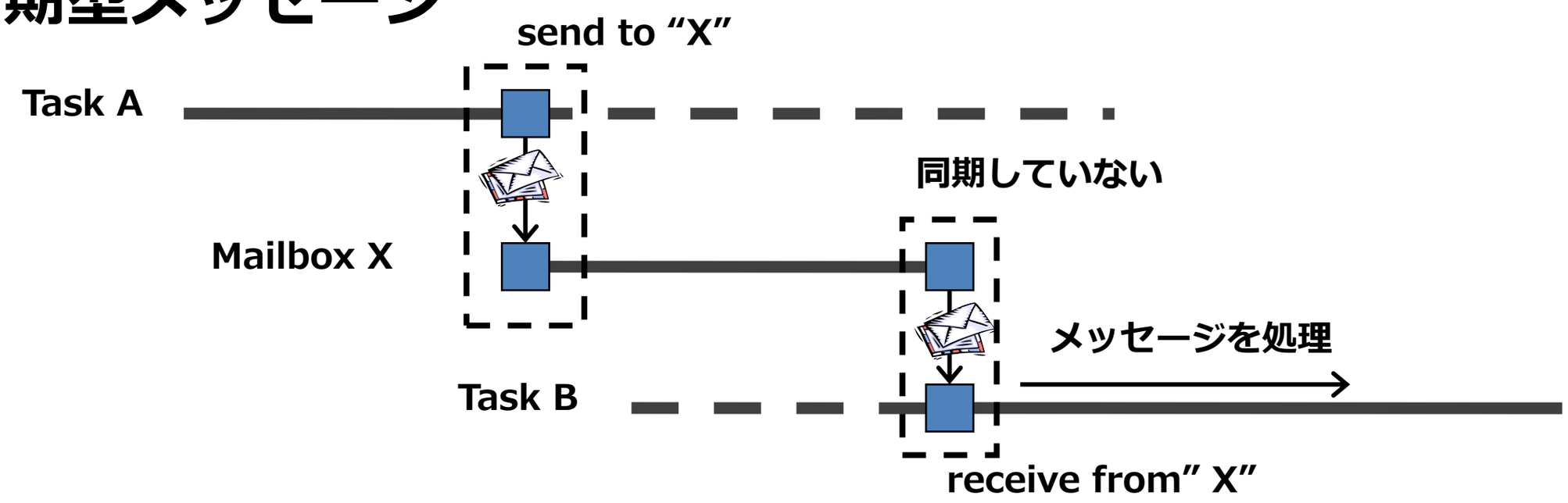
メッセージ方式

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

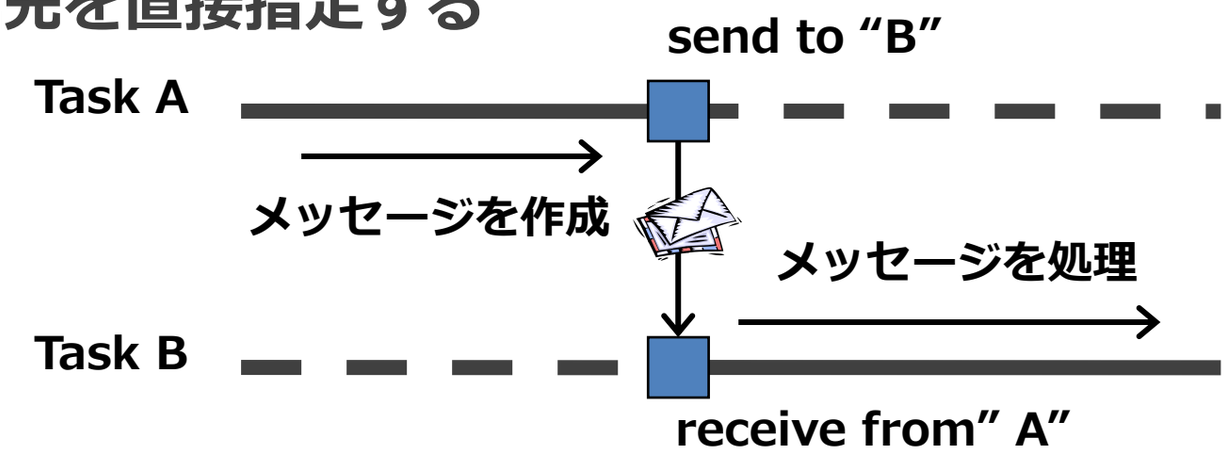


非同期型メッセージ

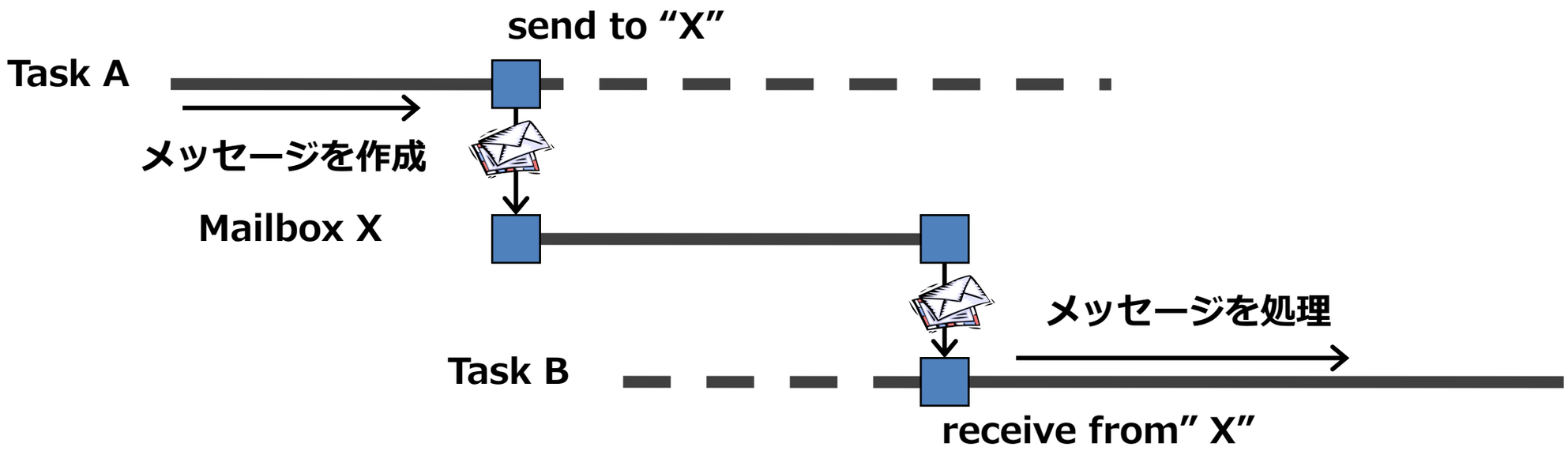


通信の方式（３）
直接通信・間接通信

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);

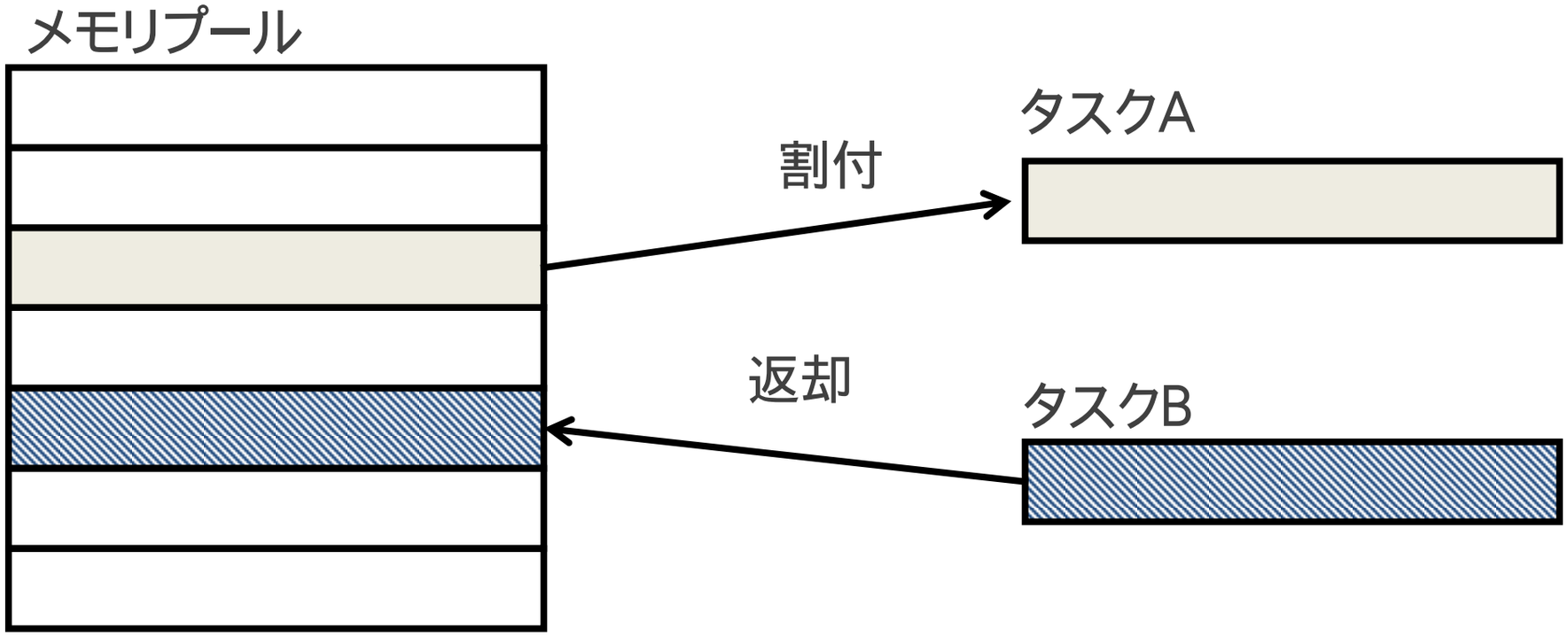
4 記憶管理

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）



5 実時間処理

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

1. 時刻の取得、設定

- 相対時間
- 絶対時間

2. 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

3. タイムアウト処理

- 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

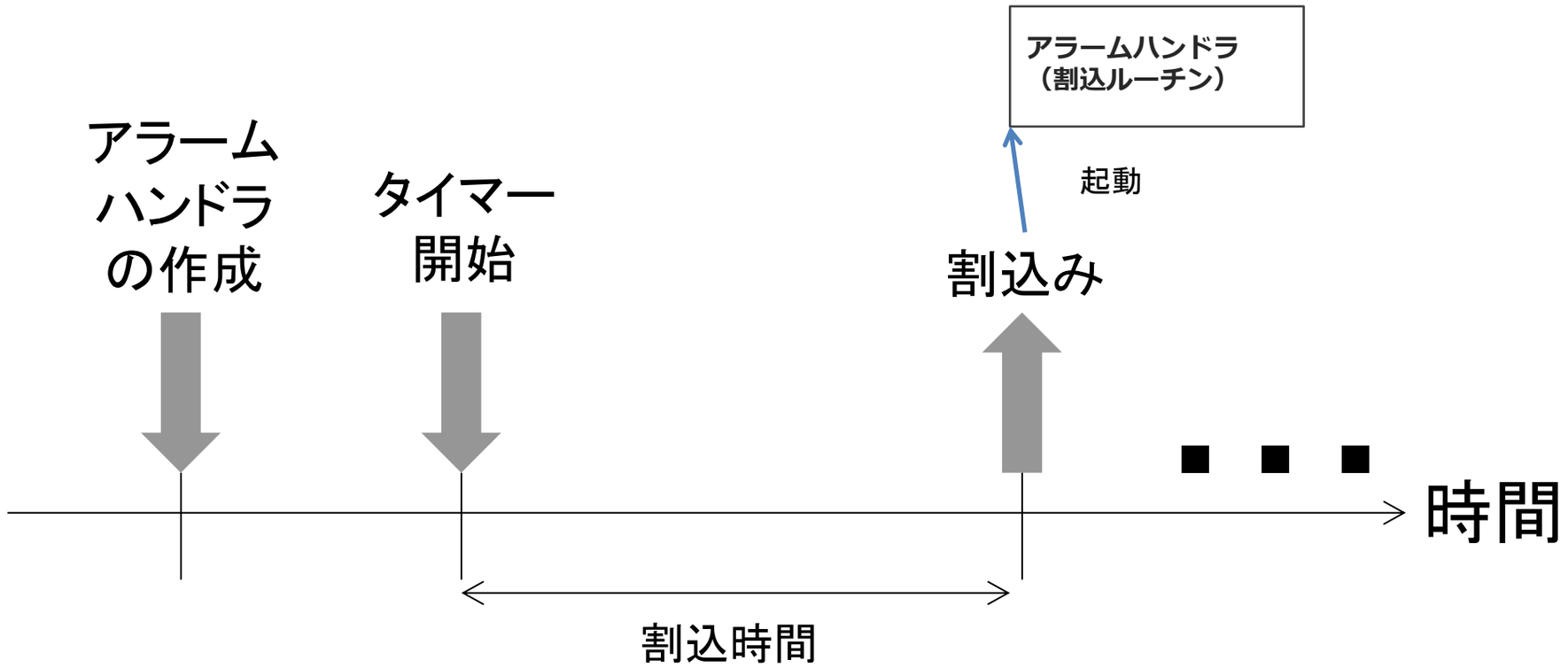
- **相対時刻**
  - OS立ち上げ時からの相対的な時刻。
  - ハードクロックのカウンタを持つハードウェアタイマを使用する。

- **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。

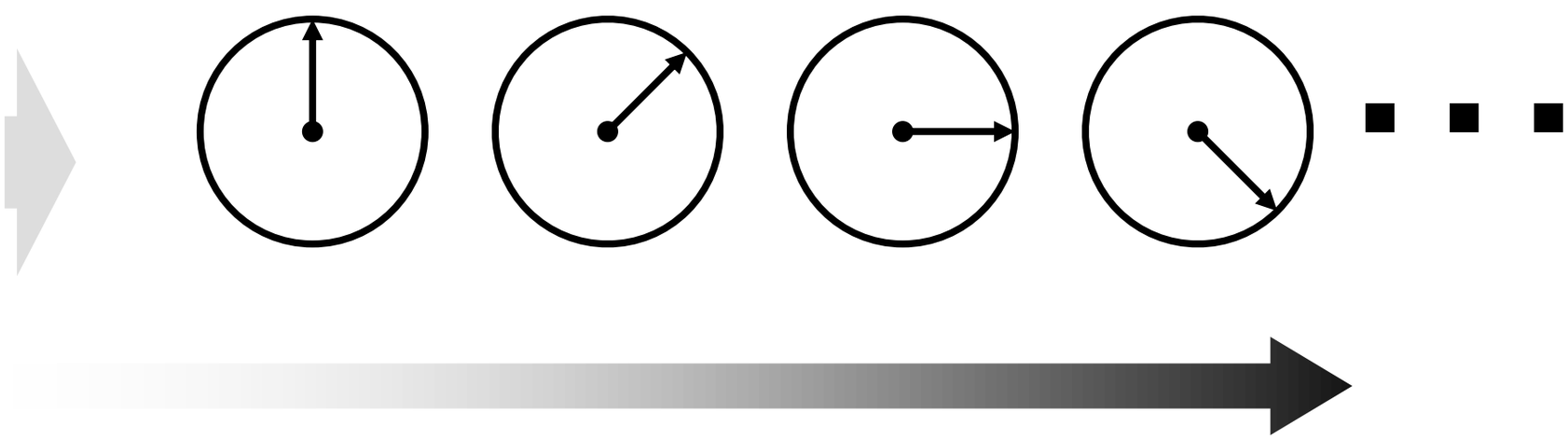
２．タイマ割込み処理

- **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。

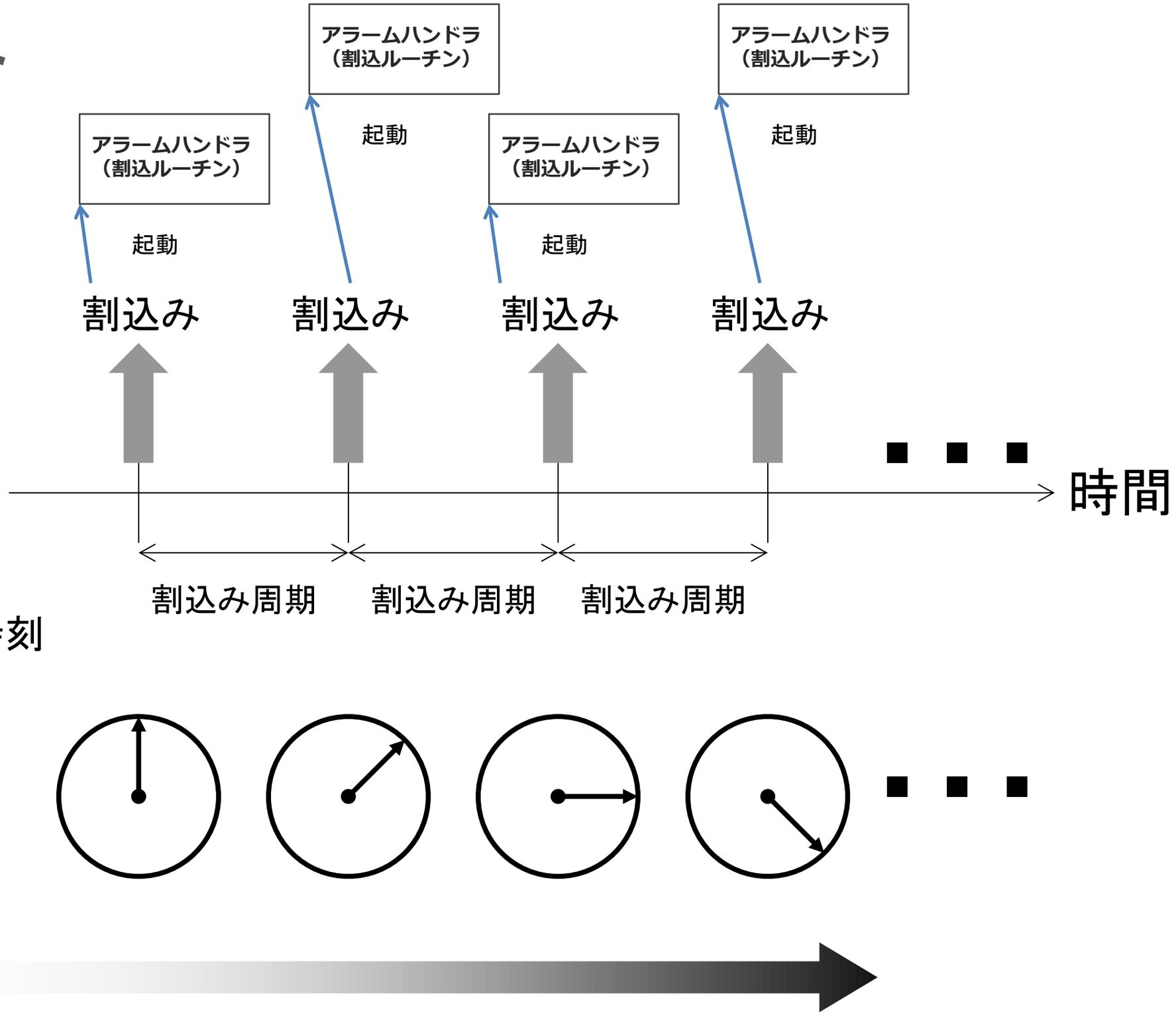
アラームの例



システム時刻



周期割込み



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。

第５章
まとめ（RTOS）

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

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）

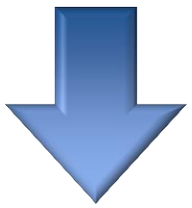
リアルタイムシステムの必要な機能とRTOS

RTOSが提供する機能

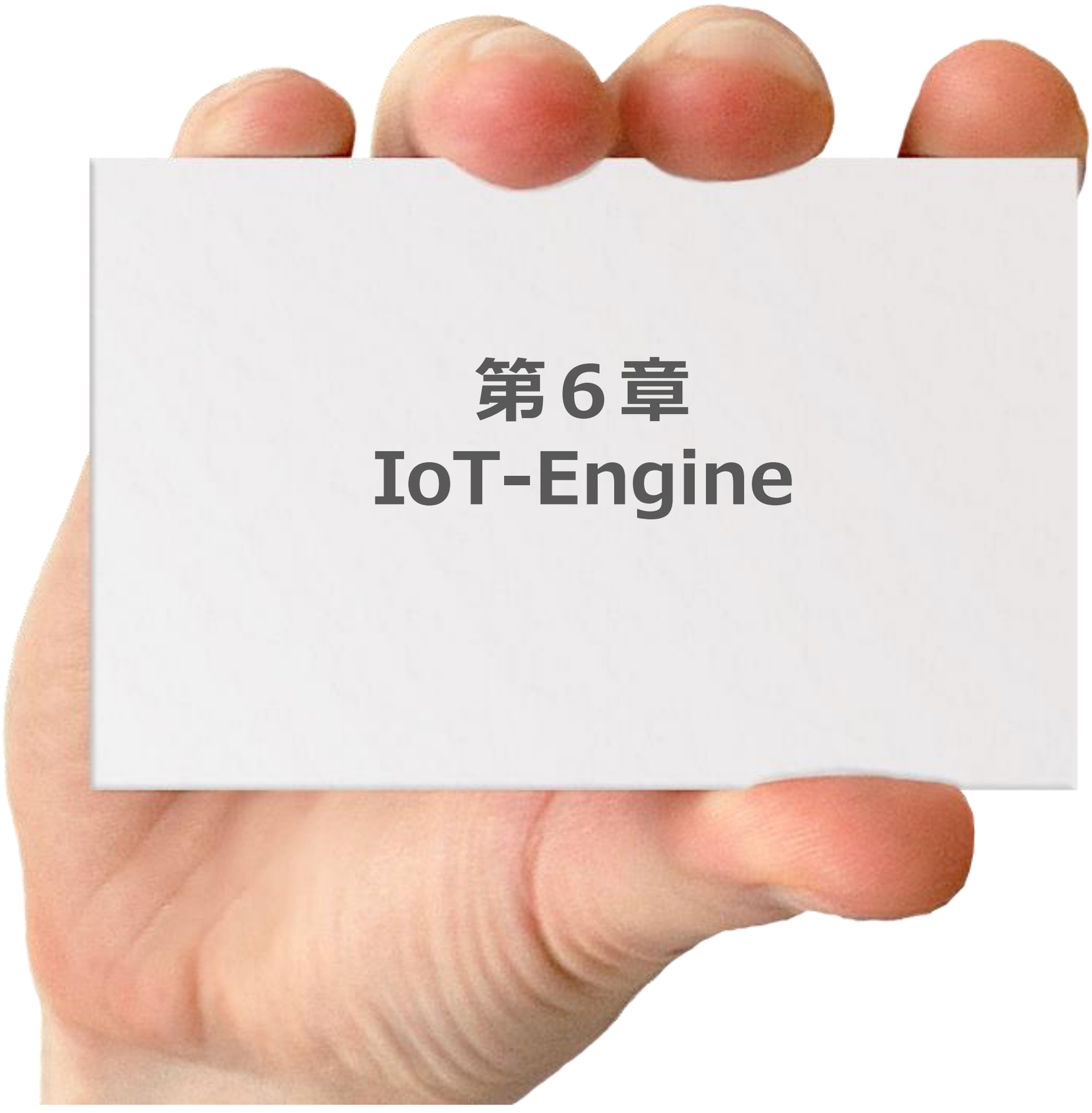
- タスク管理機能
- タスク同期管理機能
- 同期通信機能
- メモリ管理機能
- 時間管理機能
- 割込み管理機能

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

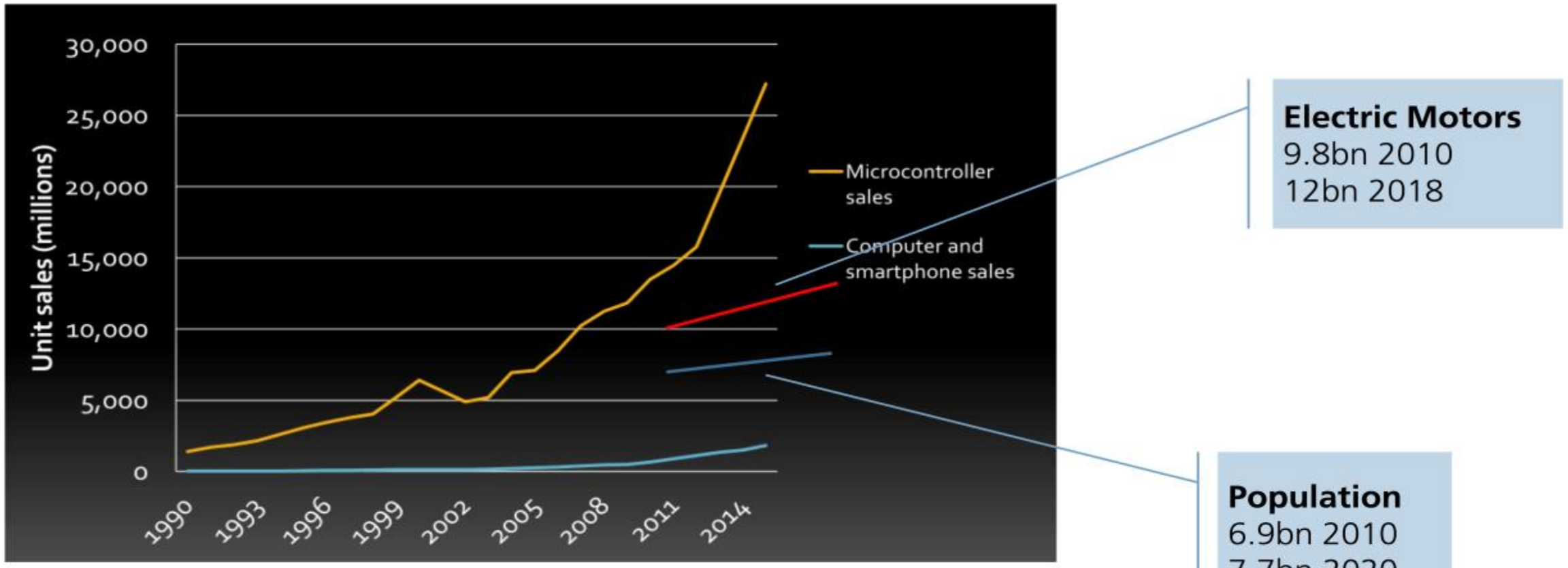


- 上位のミドルウェアでサポート



第6章
IoT-Engine

マイクロコントローラーとコンピュータの出荷数



Source: Thanki/ICSS 2013

マーケット比較（出荷数、2012）

		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	PC	355 M	
	Smartphone	712 M	
	Sub Total	1,067 M	非組込系

IC Insights: “Research Bulletin: MCU Market on Migration Path to 32-bit and ARM-based Devices”, April 25, 2013

http://www.icinsights.com/data/articles/documents/541.pdf

IDC: “Soft PC Shipment in Fourth Quarter Lead to Annual Decline as HP Holds Top Spot, According to IDC”

http://www.idc.com/getdoc.jsp?containerId=prUS23903013#.UO9hyW9QWSq

IDC: “Strong Demand for Smartphones and Heated Vendor Competition Characterize the Worldwide Mobile Phone Market at the End of 2012, IDC says”

https://www.idc.com/getdoc.jsp?containerId=prUS23916413

“Trillions of Connected Devices”
一兆個のデバイスがネット接続される

9 billion of connected devices in 2013
Gartner: 26 billion of units by 2020
Cisco: 50 billion of connected things by 20202
IDC (2013): 212 billion of connected things by 2020

分解されて、ネットワークに溶け込む組込み機器

IoT, M2M時代の組込み

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

M2M, IoT型組込みアーキテクチャ

従来型の組込機器
（血圧計）

クラウドサーバー

クラウド（インターネット）

データ送信

データ閲覧

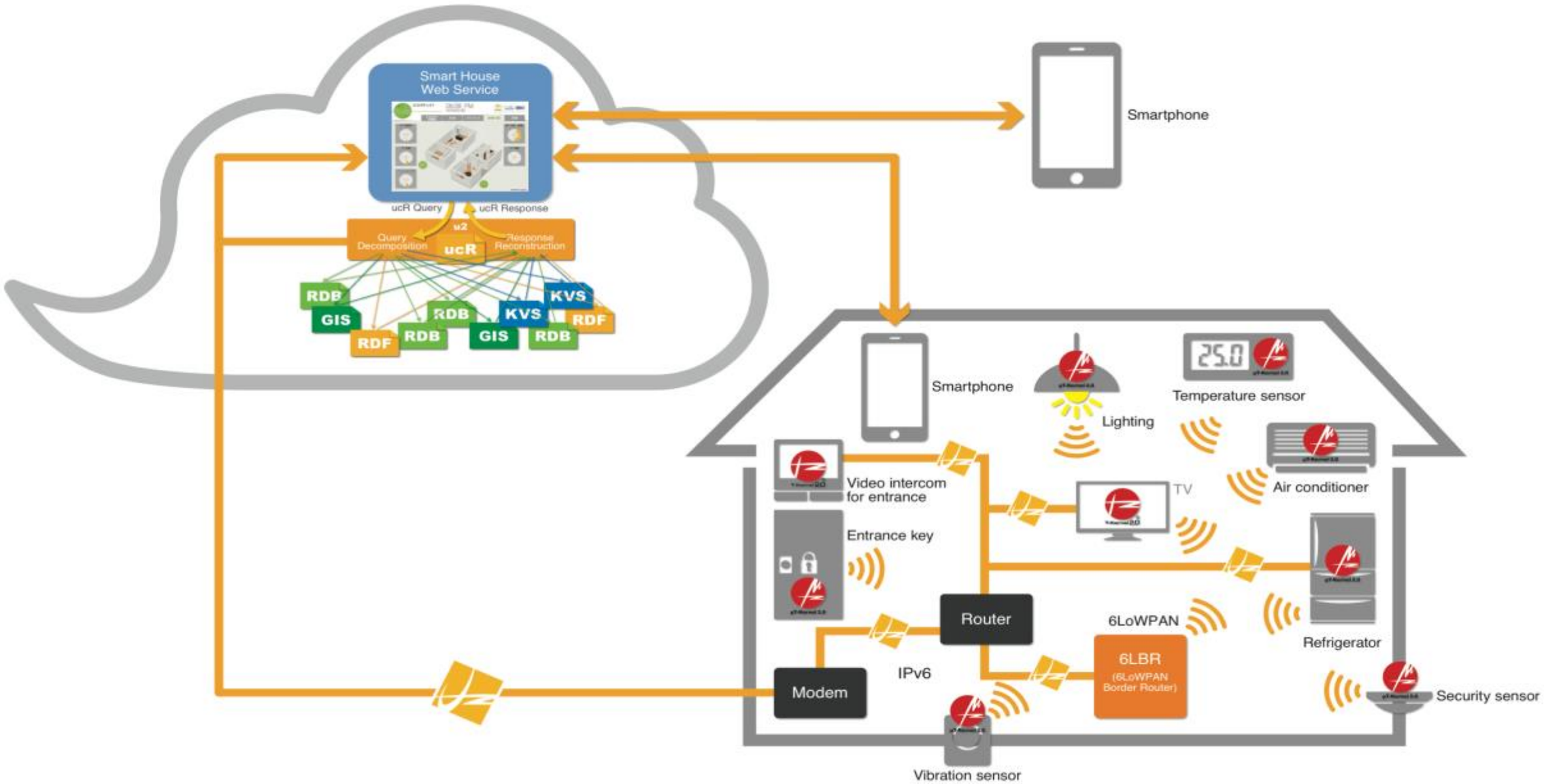
センサーとUIの分離

ネットワーク機能つき
超小型血圧センサー

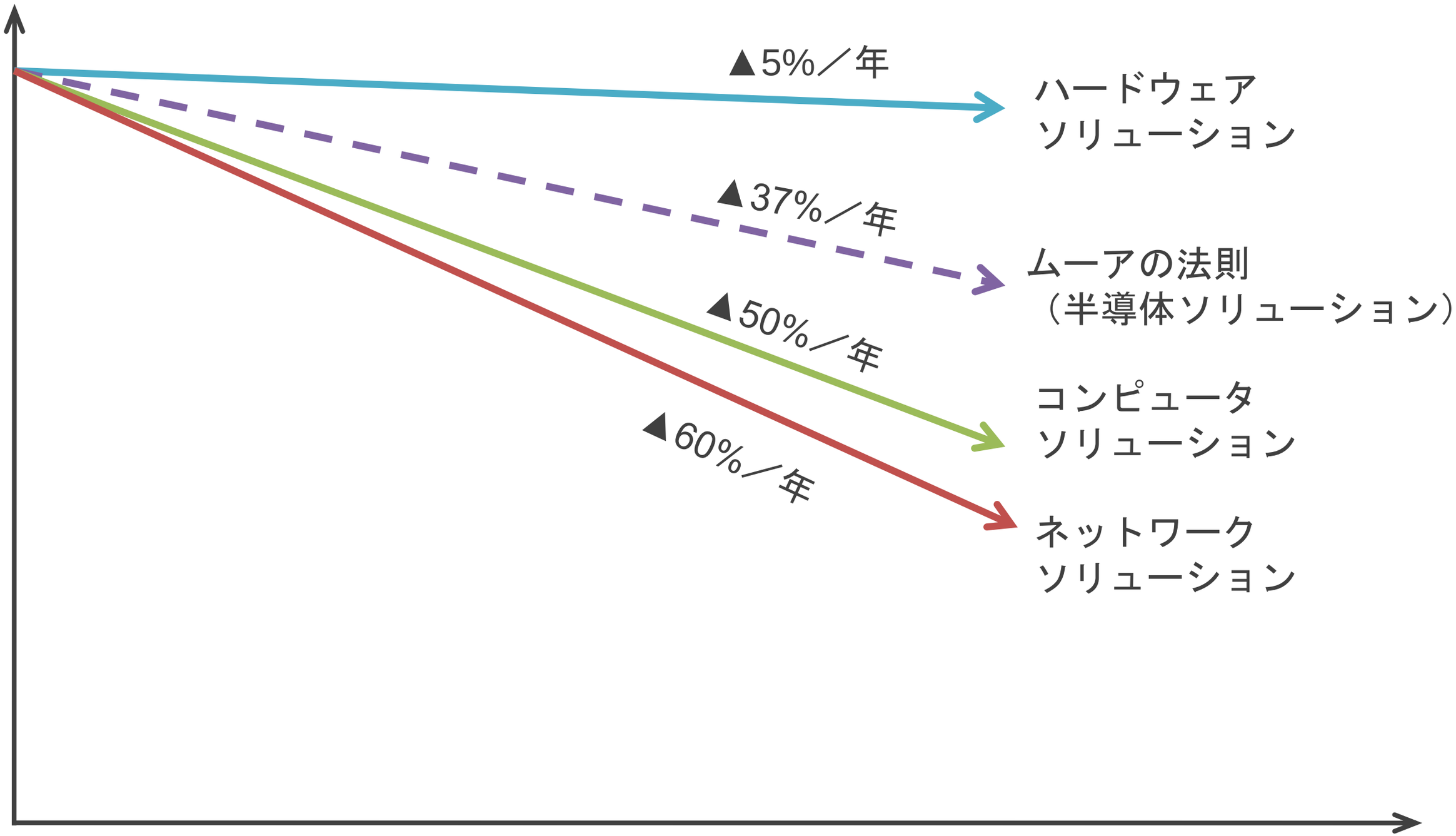
超小型組込み
機器

スーマートフォン

クラウドやスマホと一体化する家電

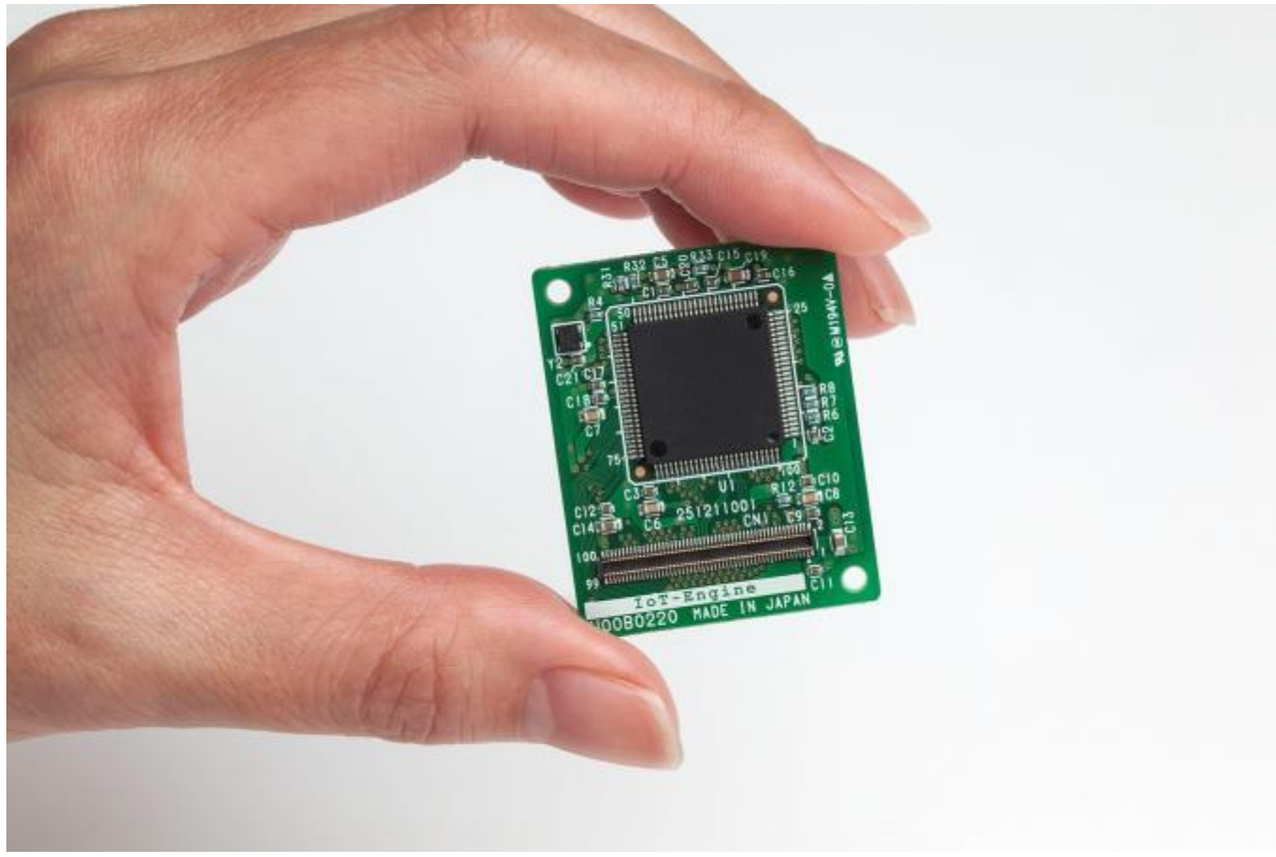


コストベネフィット



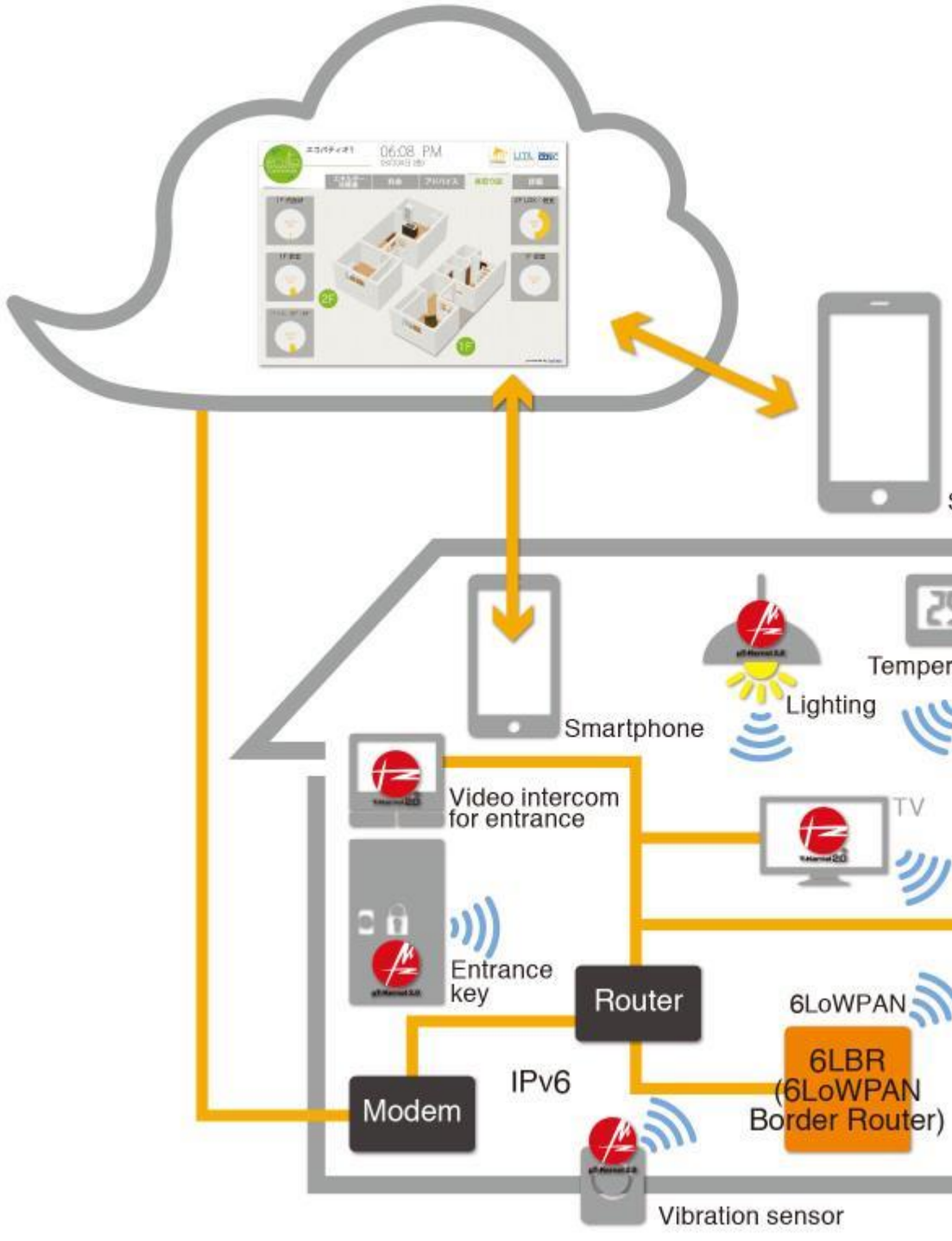
世界６カ国７社の半導体ベンダーが IoT-Engine製品化、販売へ

- **参加半導体ベンダー**
  - 東芝マイクロエレクトロニクス
  - ルネサス エレクトロニクス
  - Cypress
  - Imagination Technologies
  - Nuvoton Technology
  - NXP Semiconductors
  - STMicroelectronics



- **IoT-Engine開発キット発売**
  - パーソナルメディア
  - ユーシーテクノロジ

IoT-Engine



IoT機器、センサー等に
組み込む、コンピュータ
標準プラットフォーム

IoT-Engineの特長（１）

- **小型・低価格・低消費電力を目指したWPAN（IEEE802.15.4）無線搭載**
  - 周波数は、国により異なるが780MHz、915MHz、920MHz、950MHz、2.4GHzなど
    - WPAN : Wireless Personal Area Network 近距離無線通信
  - 電池やエネルギーハーベストで動作させるような機器にも対応できる低消費電力向け
- **CoAP、6LoWPANプロトコルを搭載**
  - 6LoWPANボーダールータ経由でクラウドに接続する
  - クラウドのWeb APIに親和性の高いCoAPを搭載する
- **Open IoT Platform接続**

IoT-Engineの特長（２）

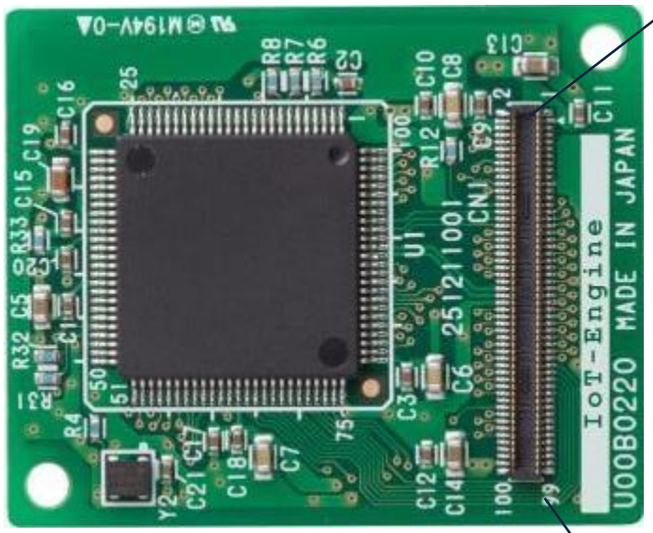
- **低消費電力対応μT-Kernel2.0 リアルタイムOS搭載**
  - マルチタスクプログラミングによる高度な制御ロジックが容易に実装できる
  - IEEE802.15.4ビーコンモードで、マイコンをDeep Sleepモードに落とせる超低消費電力対応
- **IoT-Engineのコネクタの標準化**
  - 0.4mmピッチ100ピンコネクタと、コネクタ横の固定ネジ位置
  - マイコンの違いを吸収できる自由度を持たせた信号ピン割り当て
  - 低コスト・短期開発に有効なArduino互換I/O信号ピン割り当てを持つ

IoT-Engine規格の標準化

- コネクタ、寸法規格
- コネクタ信号割当ガイドライン
- 典型的なデバイスのドライバインタフェース
- ミドルウェアインタフェース

- 標準に基づいた各種MPUのよる製品化、ミドルウェア製品化を行える仕組みを用意する。

IoT-Engineのコネクタ信号規格

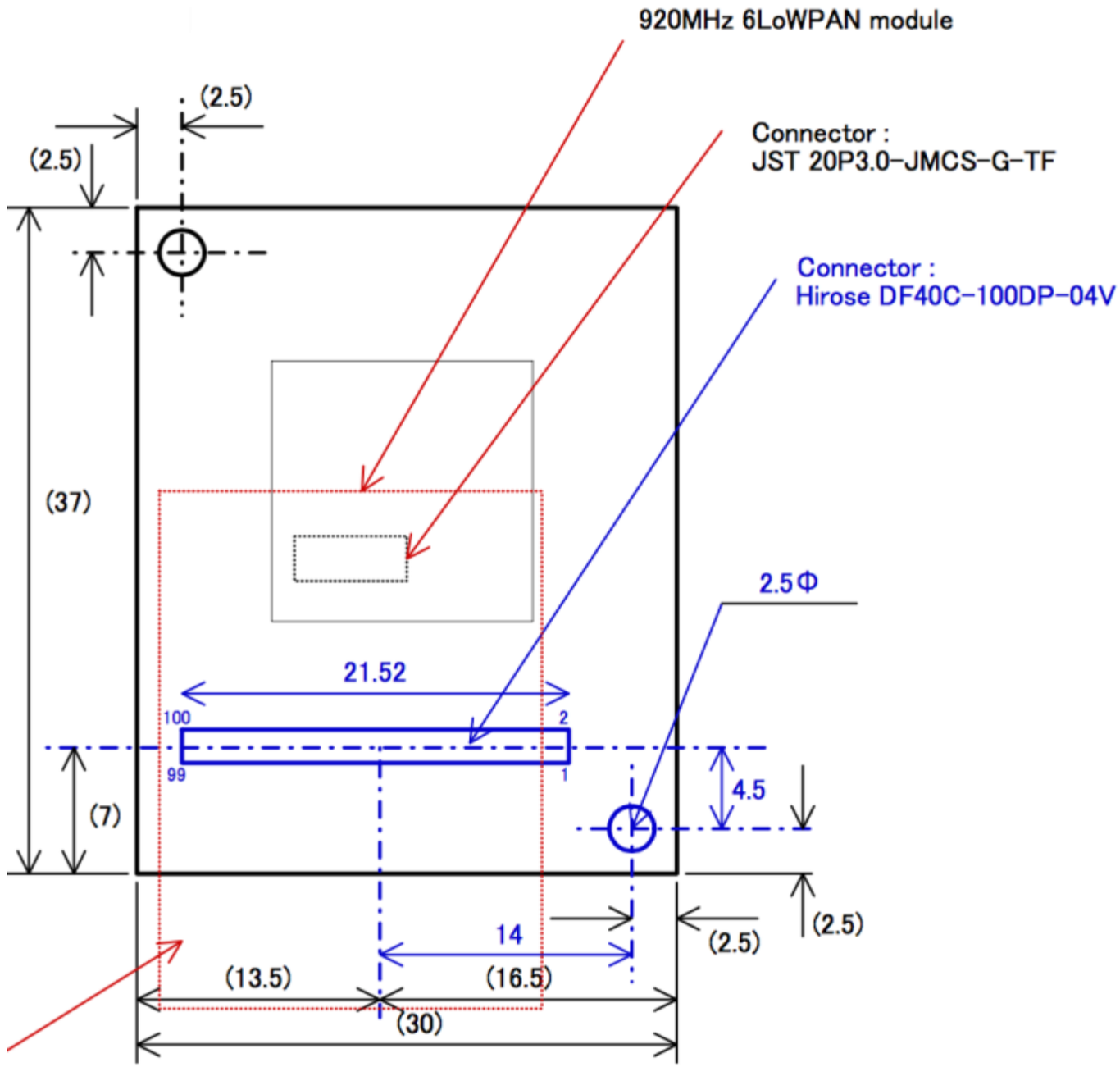


		EDGE-SIDE		OUT-SIDE	IN-SIDE			
TYPE	SIGNAL ASIGNMENT		CONNECTOR PIN#		SIGNAL ASIGNMENT		TYPE
		VBATT	1	2	D3.3V		
GPIO/INT	-WKUP	1	3	4	D3.3V		
	SCK	2	5	6	1	USER-OPT1	USER OPTION
	TXD	3	7	8	2	USER-OPT2	
	RXD	4	9	10	3	USER-OPT3	
	GPIO	5	11	12	4	USER-OPT4	
UART	GPIO	6	13	14	5	USER-OPT5	
	RXD	7	15	16	6	USER-OPT6	
	TXD	8	17	18	GND		
		GND	19	20	1	SWCLK	DEBUG
PWM	-INT	1	21	22	2	SWDIO	
	GPIO	2	23	24	3	SWO	
	PWM	3	25	26	4	Vref	
	PWM	4	27	28	5	-NMI	NMI/INT
	PWM	5	29	30	6	-INT	
	PWM	6	31	32	7	-INT	
	PWM	7	33	34	8	-INT	
	PWM	8	35	36	9	-INT	
		GND	37	38	10	-INT	
SPI	MISO	1	39	40	11	SS	SD
	MOSI	2	41	42	12	MISO	
	CLK	3	43	44	13	MOSI	
	SS	4	45	46	14	CLK	
		GND	47	48	GND		
	PWM	2	49	50	1	D+	USB
	GPIO	1	51	52	2	D-	
		GND	53	54	GND		
RTC	32kHz-IN	14	55	56	4	GPIO	GPIO
	32kHz-OUT	13	57	58	3	GPIO	
		12	59	60	2	GPIO	
		11	61	62	1	GPIO	
	-RESET_OUT	10	63	64	GND		
	-WKUP	9	65	66	8	-RESET	SYSTEM
I2C	SDA	8	67	68	7	MODE0	
	SCL	7	69	70	6	MODE1	
ANALOG	AIN	6	71	72	5	CTS	UART
	AIN	5	73	74	4	RXD	
	AIN	4	75	76	3	TXD	
	AIN	3	77	78	2	SCK	
	AIN	2	79	80	1	RTS	
	AIN	1	81	82	AGND		
		AGND	83	84	8	AI	ANALOG
RF CONTROL /DEBUG	RF_SWCLK	6	85	86	7	AI	
	RF_SWDIO	5	87	88	6	AI	
	RF_SWO	4	89	90	5	AI	
	RF_NMI	3	91	92	4	AI	
	RF_RESET	2	93	94	3	AI	
	RF_MODE	1	95	96	2	AI	
		OPEN	97	98	1	AI	
		OPEN	99	100	AVCC		

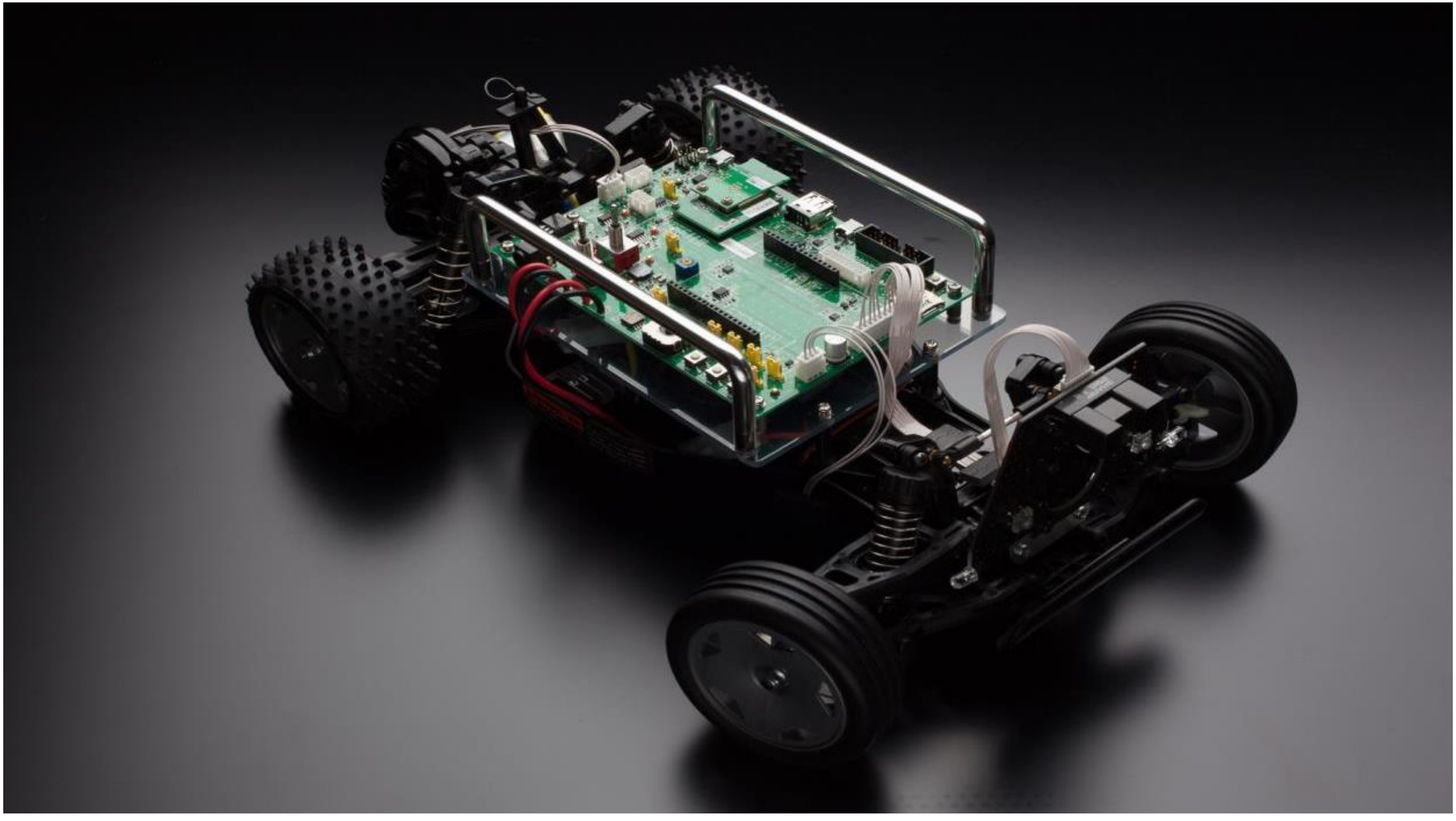
Arduino I/O connector compatible signals

機械的寸法規格

- 機械的寸法の規格化は右図青色部分
  - コネクタ
  - 固定穴
  - コネクタと固定穴の位置関係
- それ以外の寸法は参考寸法

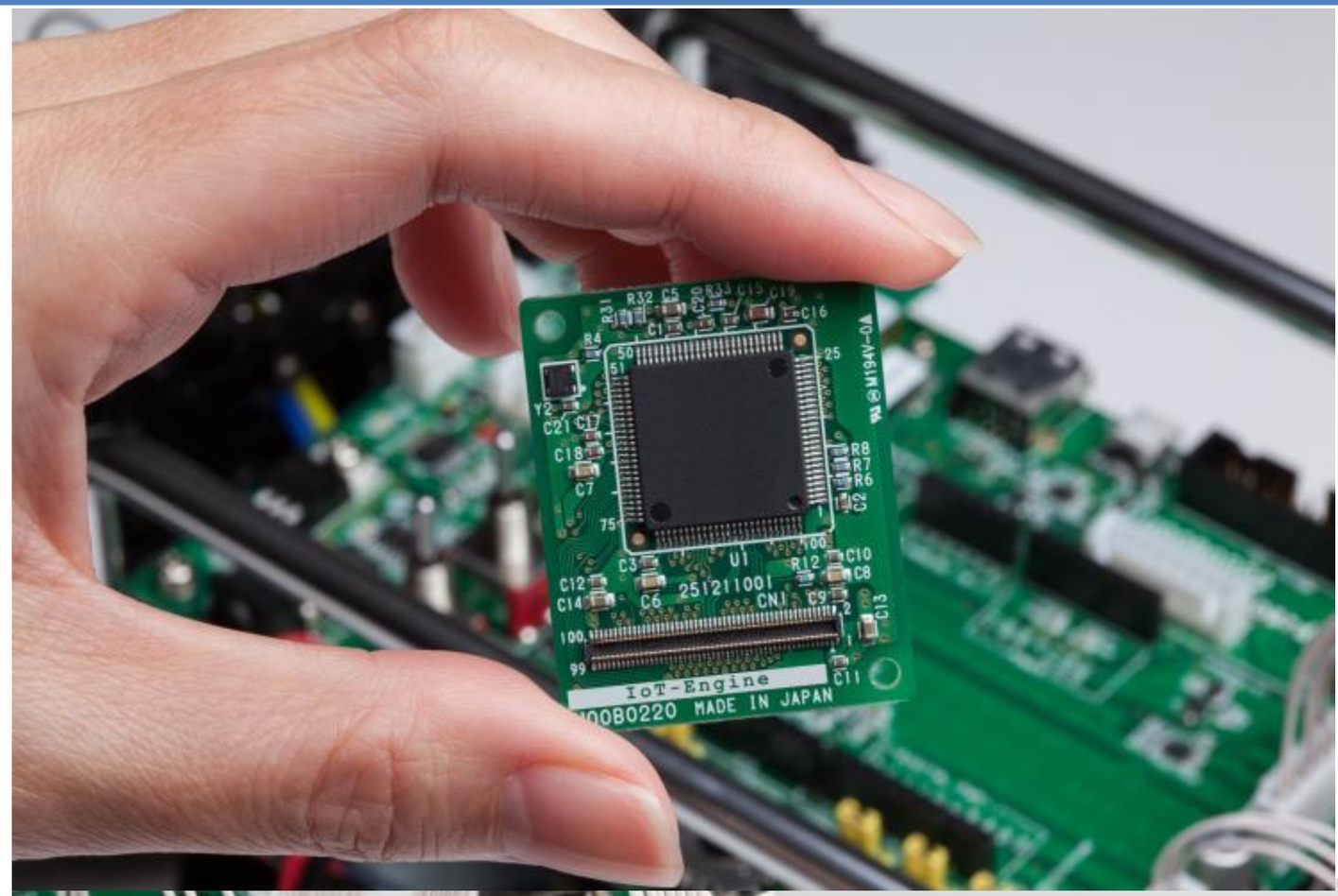


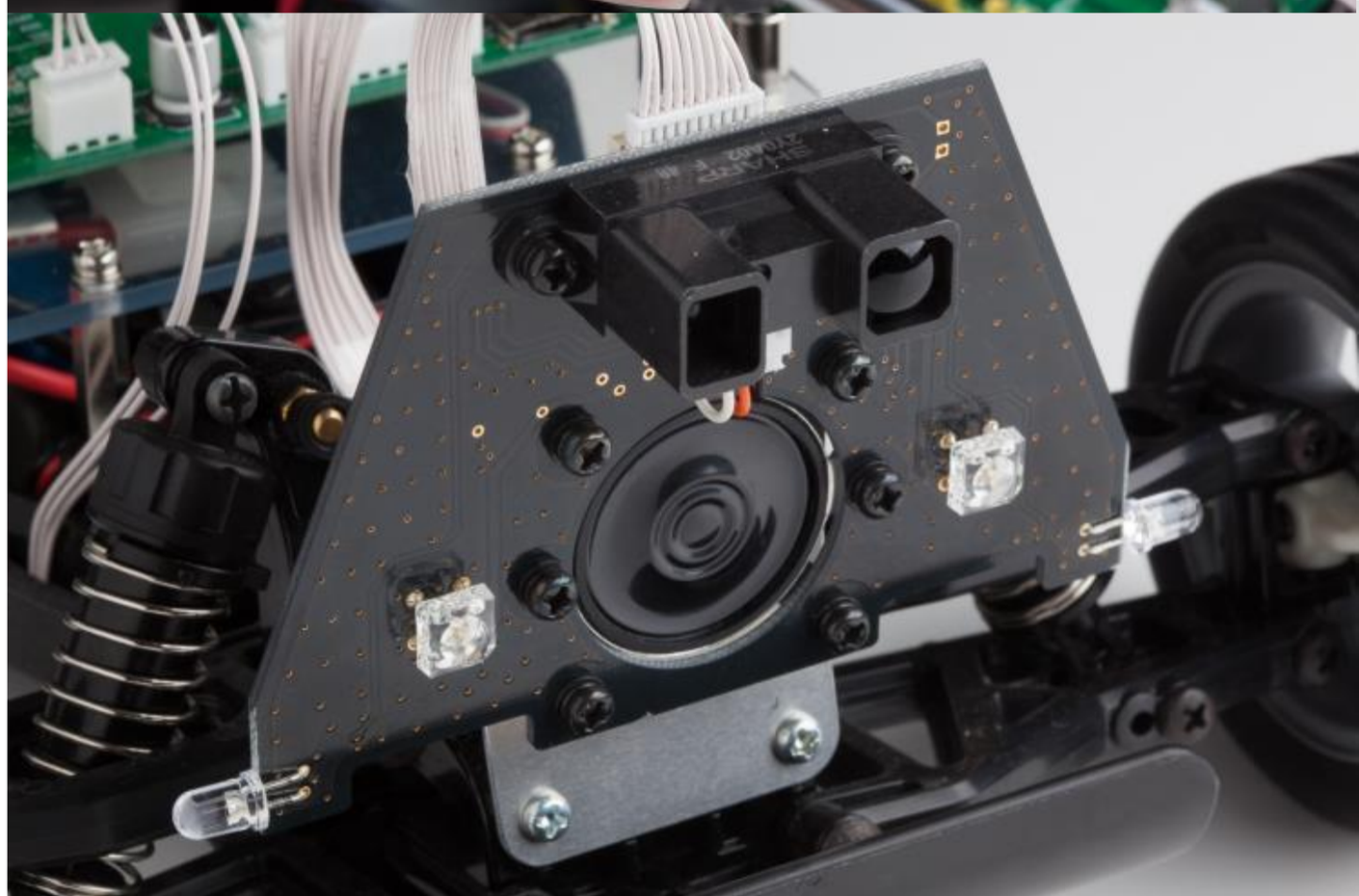
T-Car：IoT-Engine搭載　IoT教育教材



T-Car：IoT-Engine搭載 IoT教育教材

- **1/10サイズ模型自動車に、さまざまなセンサーを搭載**
  - スピードセンサー、ライントラッキングセンサー、9軸モーションセンサー、距離センサー、温度、照度・・・
  - Arduino互換I/Oコネクタを搭載し、市販パーツ（シールド）や自作基盤で拡張可能
- **UCT 6LoWPANボーダールータ（別売品）経由でクラウドに接続**
  - クラウドに接続された外部センサーからの情報を連携させてコントロール
- **プログラム開発、デバッグに便利なワークベンチ**





【講座】T-Kernel/ITRON入門テキスト「組込みリアルタイムシステム入門」

著者　YRP Ubiquitous Networking Lab.、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。