AWS Black Belt Online Seminar

# FreeRTOS
## Kernel 編

山岡 卓紀夫

Solutions Architect
2023/09

# 自己紹介

名前：山岡 卓紀夫 (やまおか たきお)

所属：

アマゾンウェブサービスジャパン合同会社
技術統括本部 西日本ソリューショングループ
ソリューションアーキテクト

経歴：

組み込みソフトウェア開発（MCU、DSP、Firmware、RTOS）
アルゴリズム研究開発（動画コーデック、センシング、CPU/GPU最適化）

好きなAWSサービス：

AWS IoT Core, FreeRTOS

# 本セミナーの対象者

本セミナーでは以下のような方を対象に、サービスを活用する上での Tips を紹介します。

- これから FreeRTOS の利用を検討しているエンジニアの方
- FreeRTOS 上でソフトウェアを開発するエンジニアの方

# アジェンダ

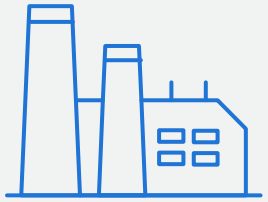
1. はじめに
2. FreeRTOSとは
3. FreeRTOS Kernel
4. FreeRTOS ライブラリ 概要 ※
5. はじめかた
6. まとめ

※ 別途ライブラリ編を実施予定

# アジェンダ

1. **はじめに**
2. FreeRTOSとは
3. FreeRTOS Kernel
4. FreeRTOS ライブラリ 概要
5. はじめかた
6. まとめ

# IoT のユースケース

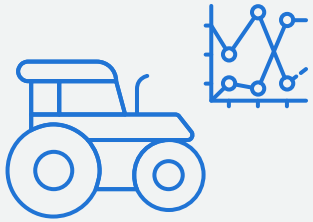


生産性とプロセス最適化

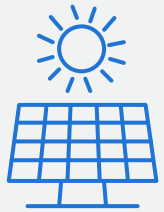
リモートで患者の健康状態を
モニタリング

在庫の可視化と
倉庫業務の最適化

家、建物、都市のスマート化



スマート農業



スマート電力

コネクテッドカー、
自動運転による輸送の変革

家庭、オフィス、工場における
安全性の向上

# IoT 利用の増加

IoT デバイスのクラウド接続と
多数のデバイス管理が一層重要に

**Total connected device growth by 2030**

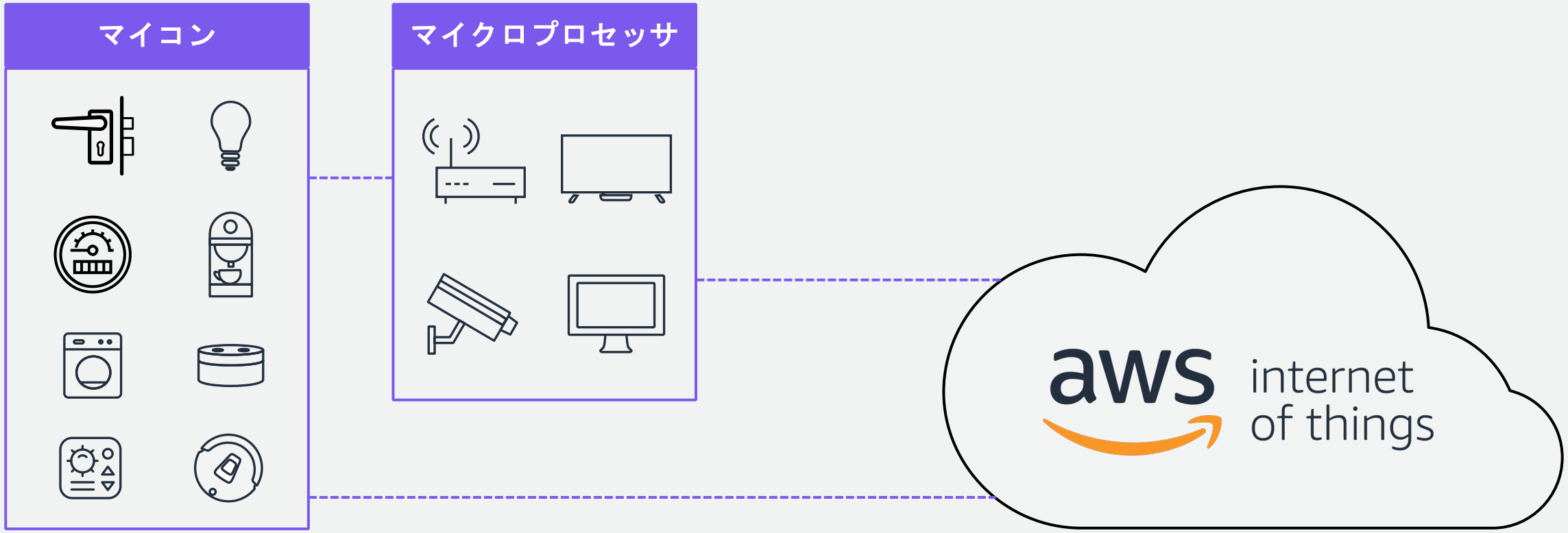
# 28.5B devices

**(285億デバイス)**

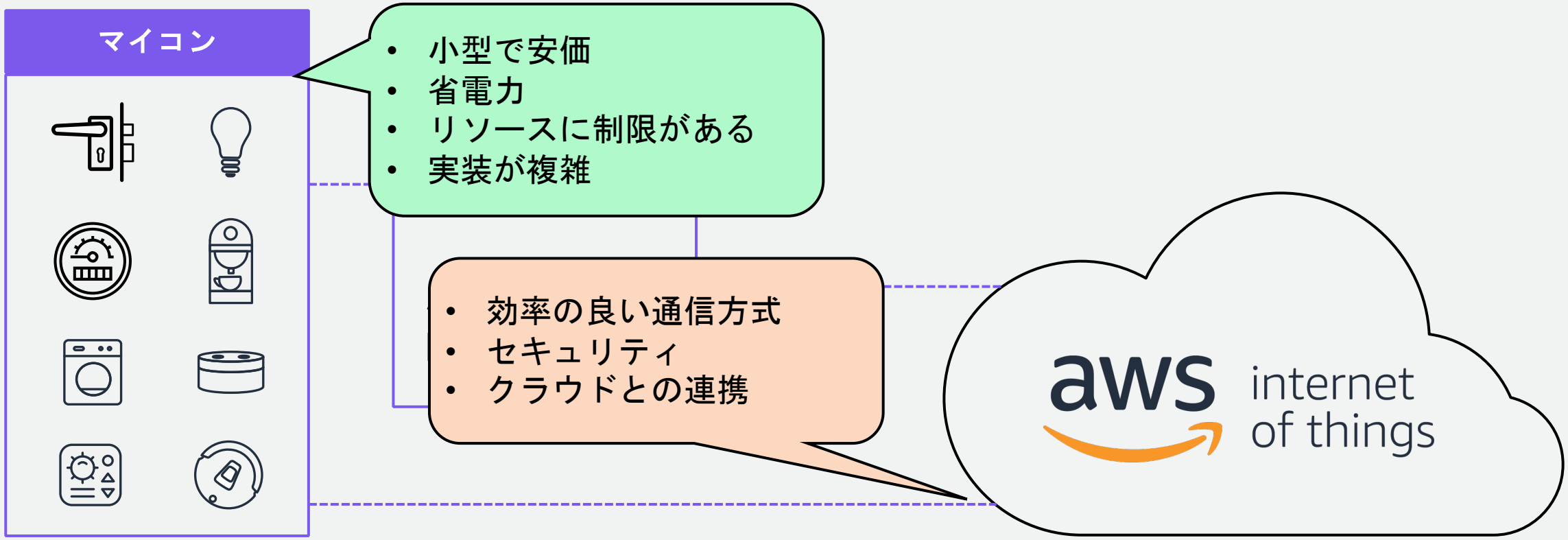
**DEVICES**

# AWS IoT と繋がるデバイスを作るには？

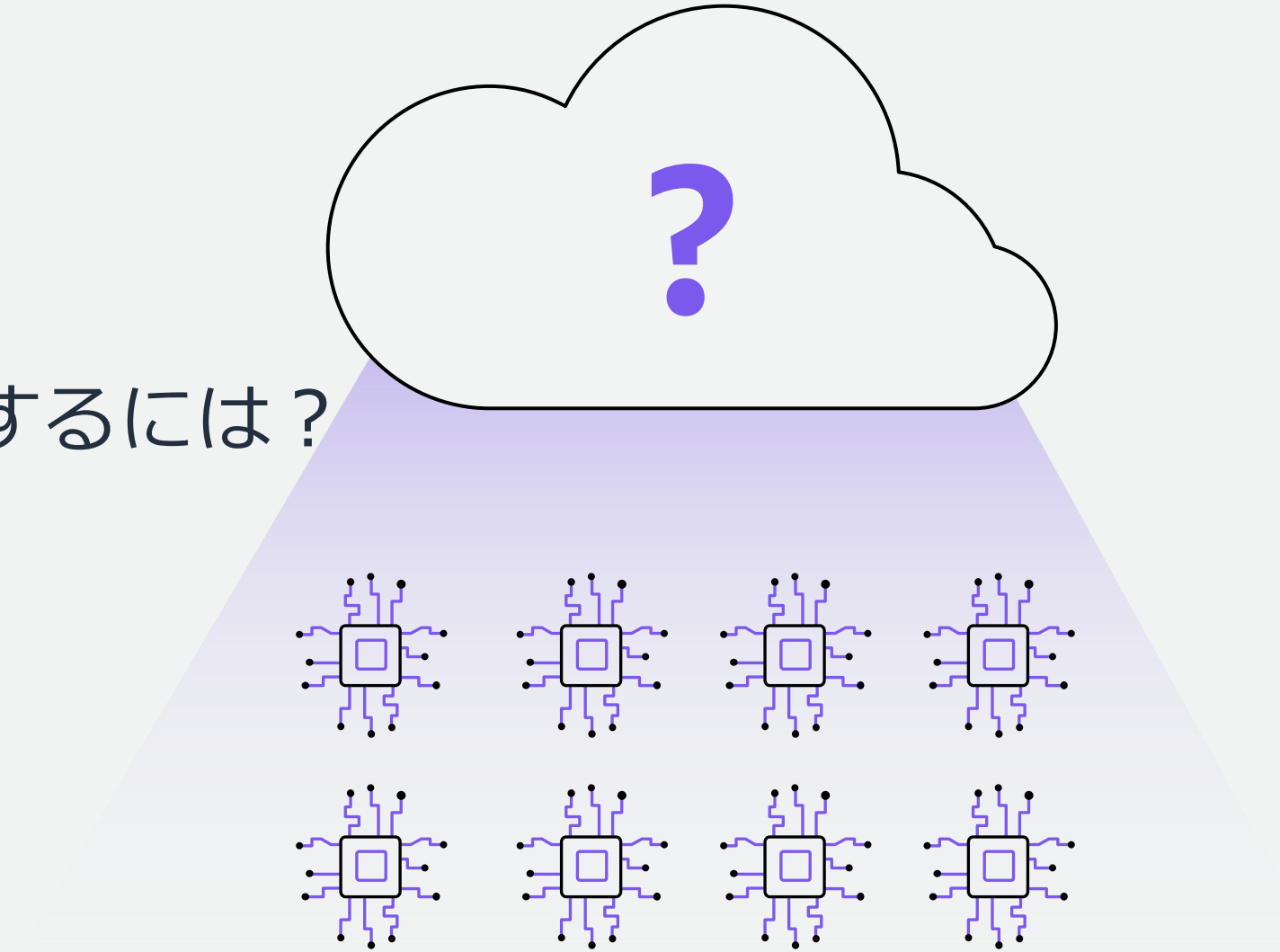
# 何億もの IoT デバイスは
# マイコンやマイクロプロセッサで動作している



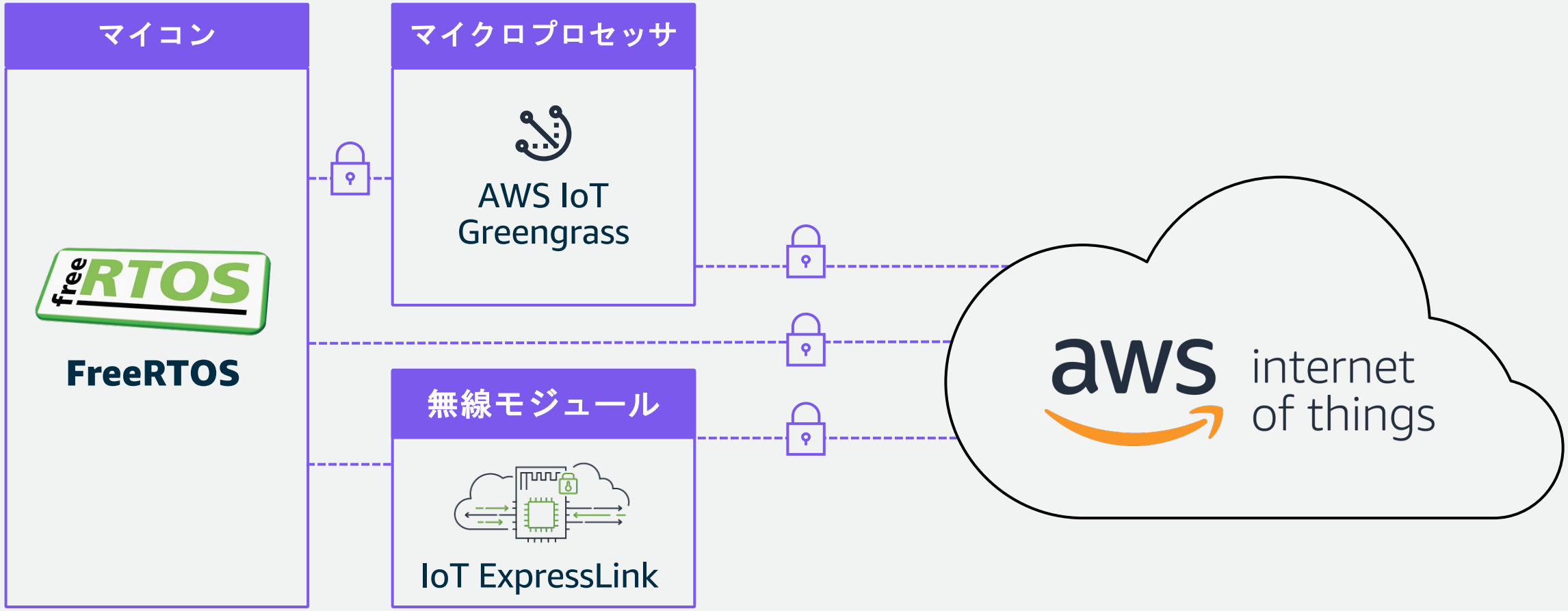
# 何億もの IoT デバイスは
# マイコンやマイクロプロセッサで動作している



# マイコンデバイスで
# AWS を簡単に使用するには？



# セキュアに AWS IoT と接続



# アジェンダ

1. はじめに
2. **FreeRTOSとは**
3. FreeRTOS Kernel
4. FreeRTOS ライブラリ 概要
5. はじめかた
6. まとめ

# FreeRTOS

20+ 年間にわたり広く配布された信頼性

RISC-V や Arm v8-M など
40以上のアーキテクチャでサポート

広範なエコシステム

MIT ライセンスの無償オープンソース

IoT デバイスに必要な機能をサポートする
豊富なライブラリ群

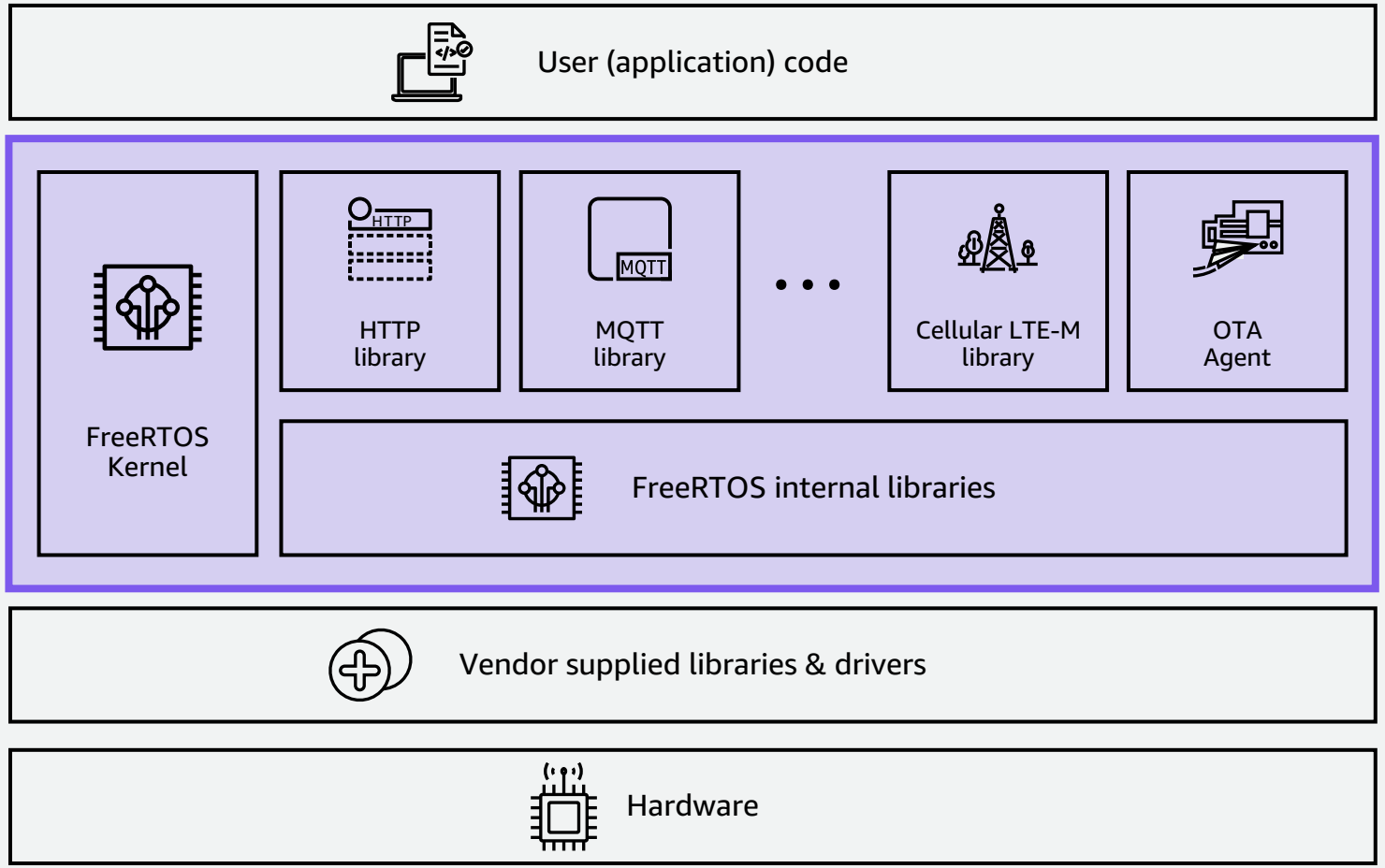
最大 2 年間の安定バージョン (LTS) や
加えて最大 10 年間の追加メンテナンスプラン 提供


https://aws.amazon.com/freertos/
https://freertos.org

# FreeRTOS System Stack

## 下記で構成

- FreeRTOS Kernel
- FreeRTOS ライブラリ



# FreeRTOS の特徴


**FreeRTOS Kernel**

リアルタイム OS (RTOS)

**Local Connectivity**

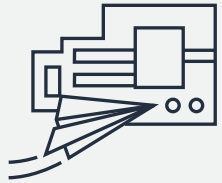
クラウド接続無しで AWS IoT Greengrass デバイスと通信

**Cloud Connectivity**

マイコンデバイスから簡単にデータを収集、アクションを実行

**Security**

デバイスのデータと接続をセキュアに



**OTA and Code Signing**

セキュリティ更新やバグ修正などのファームウェアをデプロイ

**FreeRTOS Long Term Support**

# FreeRTOS 長期サポート (LTS)

**FreeRTOS LTS ライブラリを含めた**
**機能の安定性を2年間サポート**

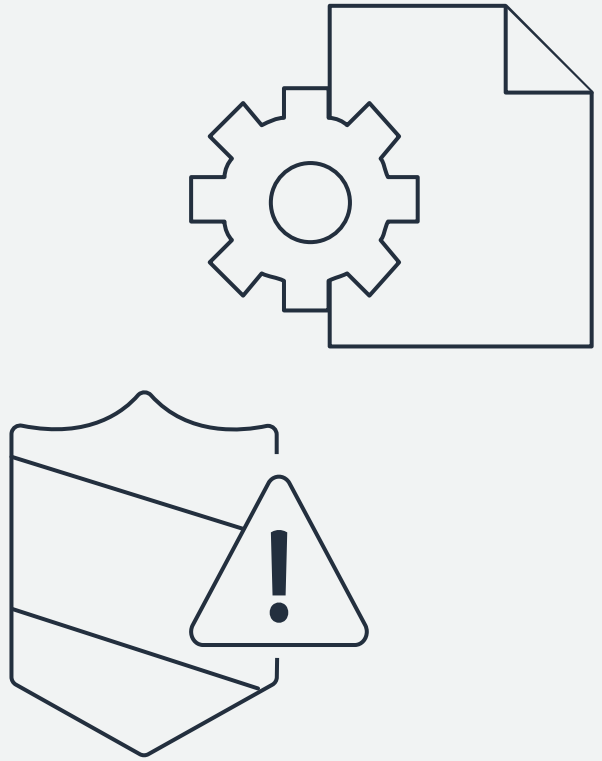
**対象:**

- FreeRTOS kernel
- Connectivity/other libraries: FreeRTOS+TCP, oreMQTT, coreHTTP, coreJSON
- Security library: corePKCS11
- AWS library: AWS IoT Device Shadow, AWS IoT Jobs, AWS IoT OTA, AWS IoT Device Defender

2年間のセキュリティアップデートや
致命的なバグ修正 (機能追加は行わない)

AWS がメンテナンスし、約1年半事にリリース

MIT ライセンスで提供されるオープンソース



https://www.freertos.org/lts-libraries.html
https://aws.amazon.com/freertos/features#FreeRTOS_Long_Term_Support

# FreeRTOS 拡張メンテナンスプラン (EMP)

**セキュリティパッチや致命的なバグ修正を**
**LTS 期間終了時点から最長10年間サポート**

製品責任リスクの軽減

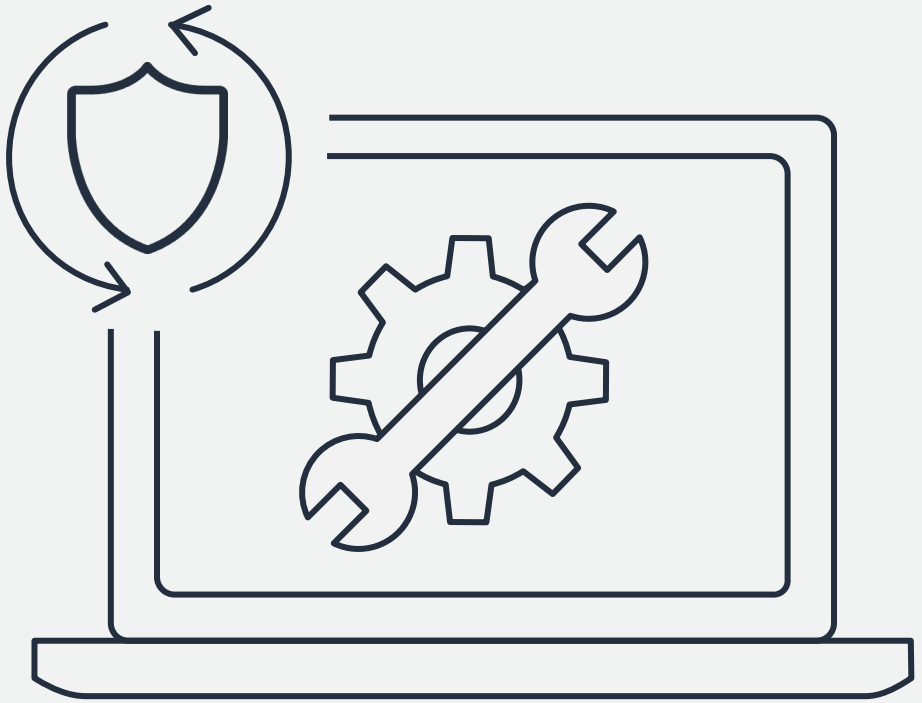
RTOS アップグレードコスト削減

長期間のデバイス セキュリティ向上

パッチ提供に関する情報をタイムリーに通知する事で
パッチを適用する遅延を削減

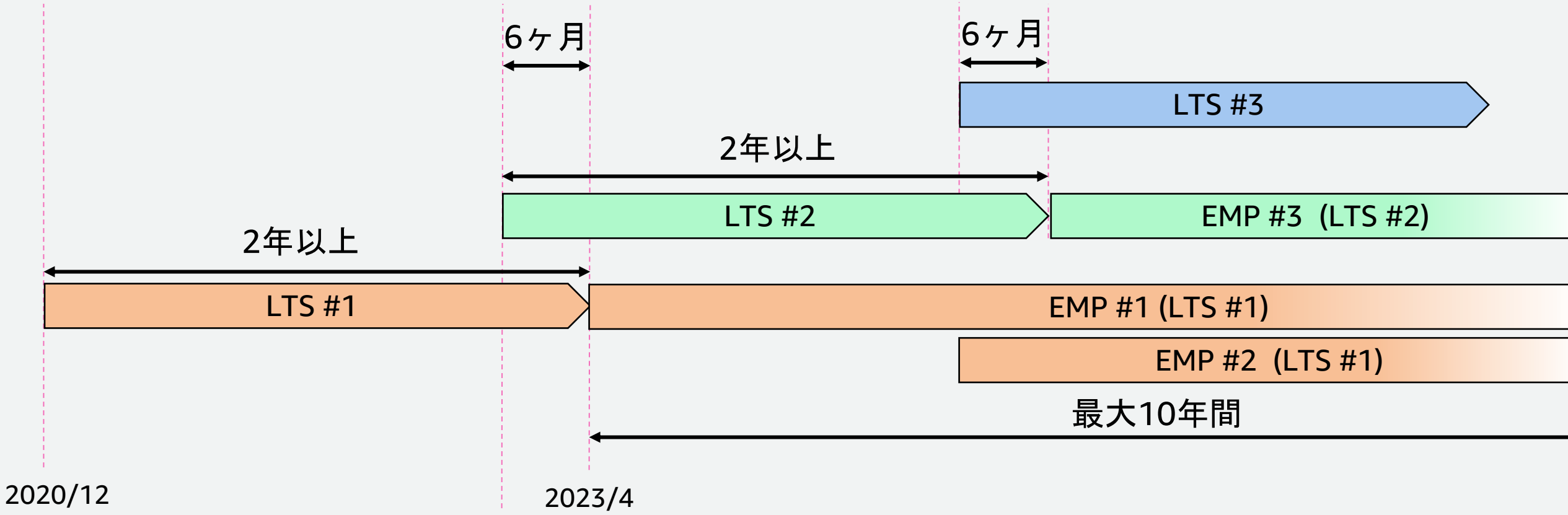
各 LTS バージョンと利用プロダクト数に応じた
年間サブスクリプション プラン

最大10年間、毎年サブスクリプションを更新



https://www.freertos.org/EMP.html
https://aws.amazon.com/freertos/features#FreeRTOS_Long_Term_Support
https://aws.amazon.com/freertos/pricing/#FreeRTOS_Extended_Maintenance_Plan_Pricing

# FreeRTOS LTS and EMP




https://aws.amazon.com/freertos/pricing/#FreeRTOS_Extended_Maintenance_Plan_Pricing

# アジェンダ

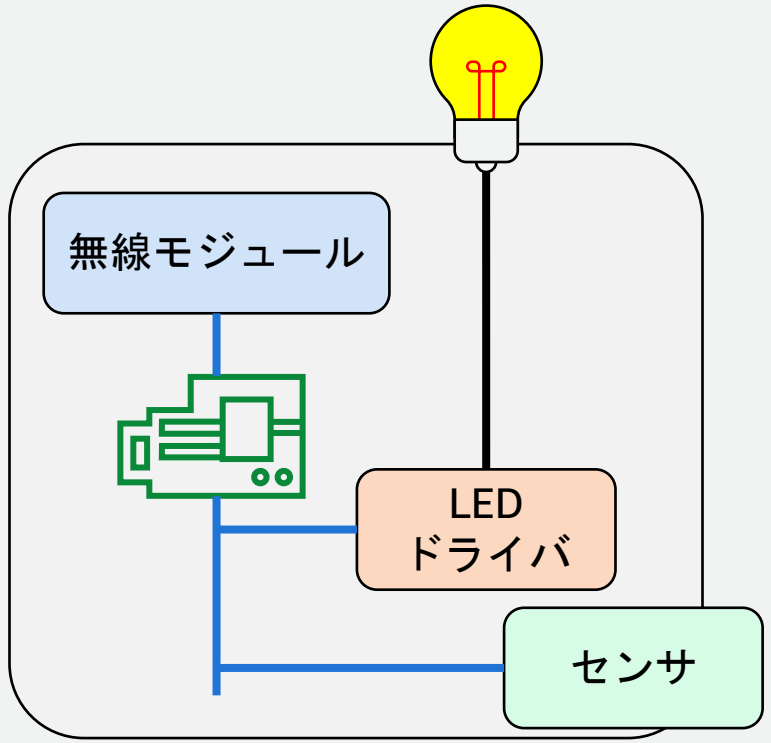
1. はじめに
2. FreeRTOSとは
3. **FreeRTOS Kernel**
4. FreeRTOS ライブラリ 概要
5. はじめかた
6. まとめ

# なぜ リアルタイム OS (Kernel) が必要か？

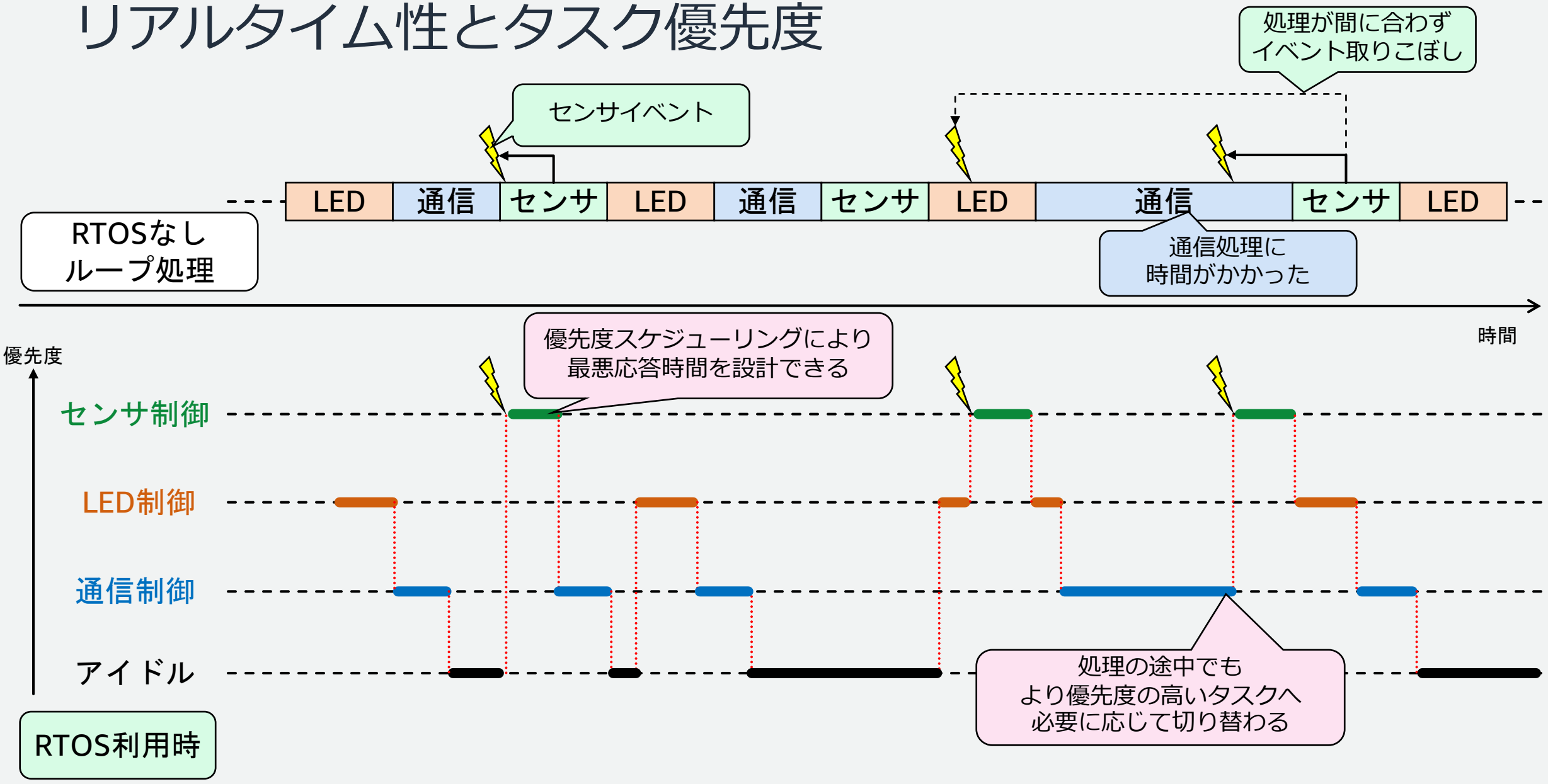
エッジデバイス上では、処理するタスクが複数存在する

例) スマート電球

- LED 制御
- 人感センサ制御
- 通信制御



# リアルタイム性とタスク優先度



# FreeRTOS Kernel の特徴

- 豊富な機能群と活発な開発
- 多数のアーキテクチャをサポート
- 充実したドキュメントとサンプルコード
- オープンソース
  - 商用でも使える MIT ライセンス
  - MISRAコーディング標準ガイドラインに準拠
- 省ROM/RAM、低オーバーヘッド
  - シンプルかつコンフィギュアラブルなKernel設計
  - Kernel バイナリサイズ: 約 6KB ~ 12KB

https://freertos.org/RTOS.html

# FreeRTOS Kernel が提供する機能

## タスク

- プログラムの実行単位
- 優先度によるタスク・スケジューリング

## タスク間通信

- タスク間や割り込みハンドラからの、通知や同期、データ通信

## ソフトウェアタイマー

- 指定タイミングで Timer Service Task からコールバックを実行する

## Kernel コンフィグレーション

- Kernelが提供する各機能を調整

# FreeRTOS を使ったプログラムの流れ

## main 関数内で初期化

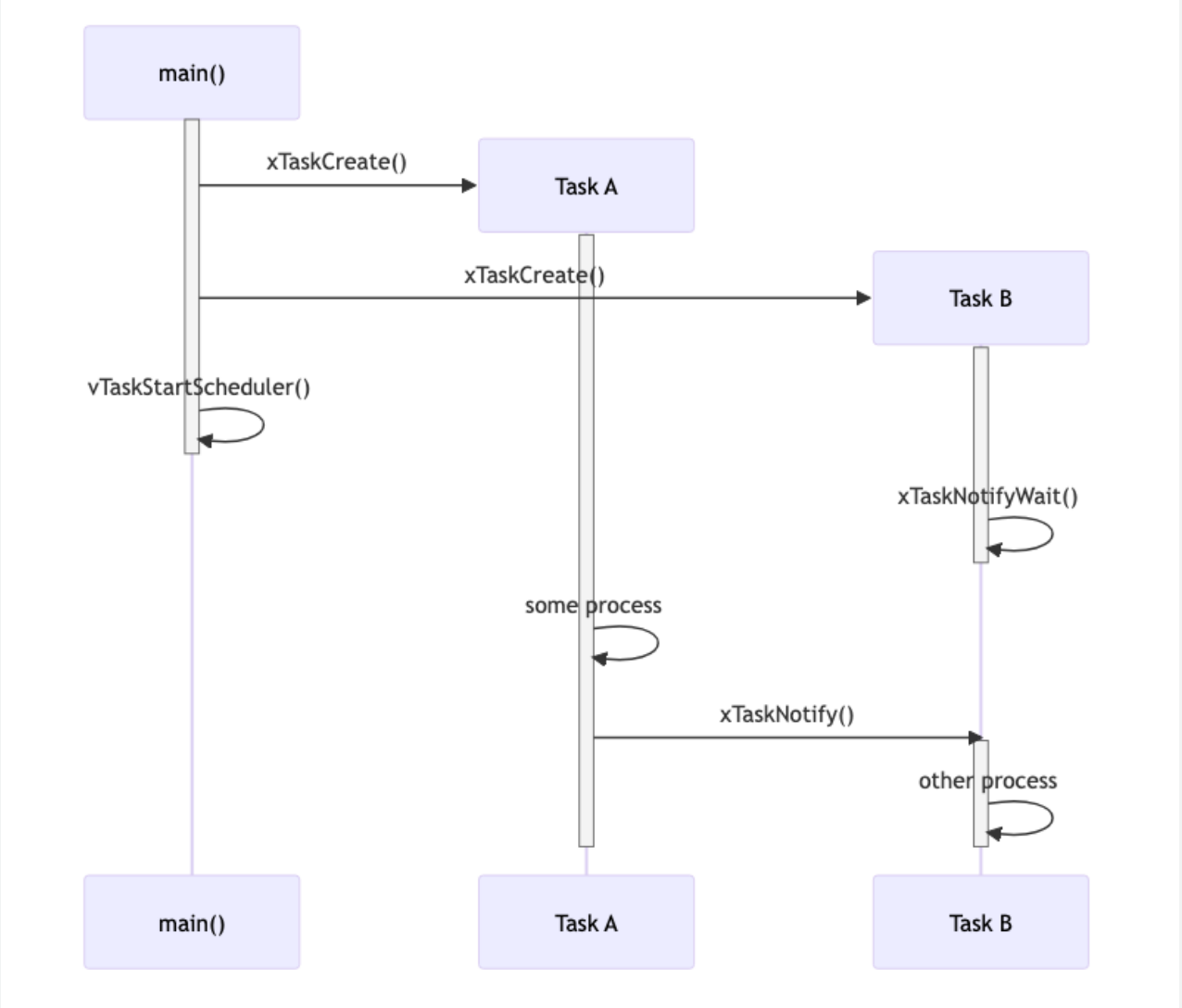
- オブジェクトの作成
  - タスク、Queueなど

## スケジューラー起動

- 作成した各タスクが起動
- 各タスクの優先度や状態に応じて
  プロセッサを割り当て

## タスク内で任意の処理

- 任意の処理を行いつつ、
  タスク間通信を利用して連携

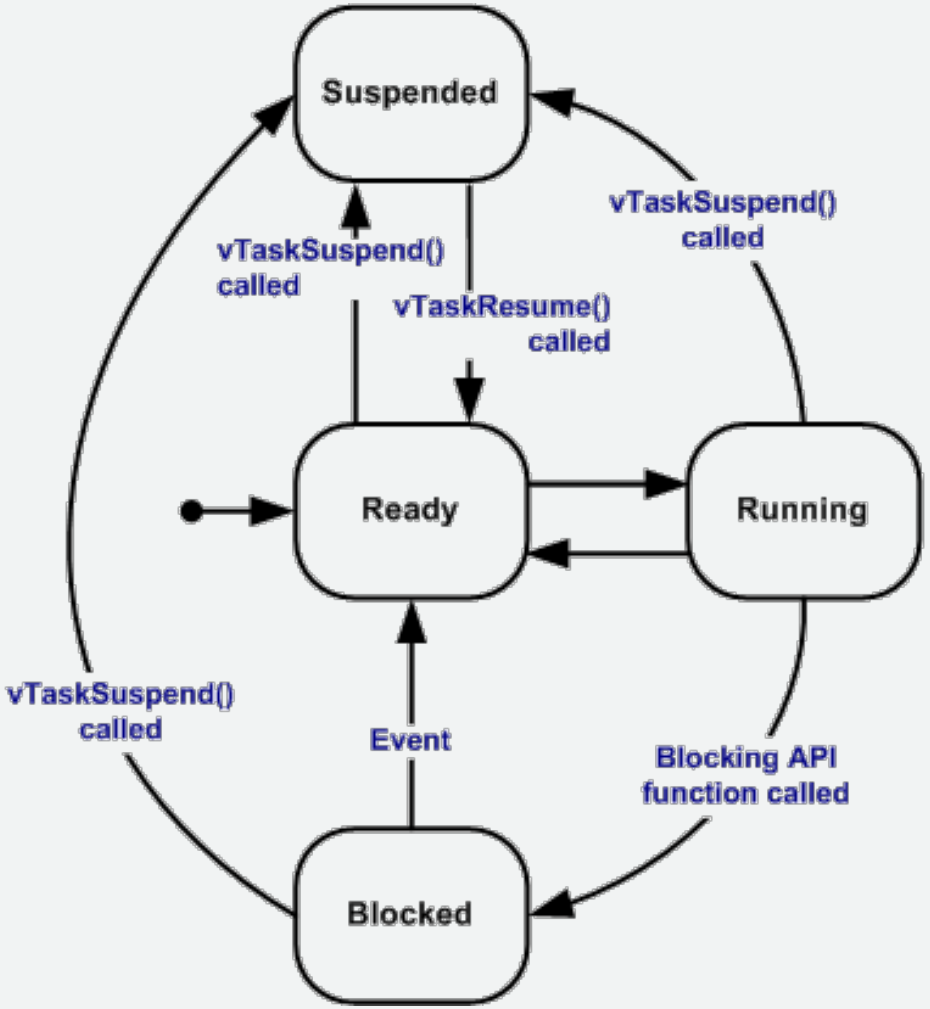


# タスク

# タスク

プログラムの実行単位

- 各タスク毎に状態を保持
  - 優先度 (0:最低優先度)
  - 実行状態 (右図は状態遷移図)
    - Running時のみプロセッサで実行
  - スタック領域
    - タスク生成時にスタックサイズを指定
  - プログラムカウンタ
  - 通知状態と通知データの配列



# タスク 主要 API

- xTaskCreate()
- vTaskDelete()
- vTaskDelay()
- vTaskStartScheduler()

https://freertos.org/a00019.html
https://freertos.org/a00112.html
https://freertos.org/a00021.html

# タスク作成例

```
void main( void )
{
     BaseType_t xReturned;
     TaskHandle_t xHandle = NULL;

     /* Create the task, storing the handle. */
     xReturned = xTaskCreate(
          vTask1,              /* Task function */
          "NAME",              /* Text name */
          STACK_SIZE,          /* Stack size in words */
          ( void * ) 500,      /* Parameter passed to the task. */
          tskIDLE_PRIORITY,    /* Priority */
          &xHandle );          /* Task handle */
     // Start the real time scheduler.
     vTaskStartScheduler();
}
```

```
void vTask1( void *pvParameters )
{
     /* Cast parameter passed to the task. */
     uint32_t uMS = ( uint32_t ) pvParameters;

     for ( ;; )
     {
          doSomething();
          /* Block for 500ms. */
          vTaskDelay( pdMS_TO_TICKS( uMS ) );
     }
}
```

# タスク間通信

# タスク間通信

## タスクやISRとの間で、通知や同期、データ送受信を行える

| 機能名 | 説明 |
| --- | --- |
| Task Notification | タスクへの通知 |
| Queue | スレッドセーフな FIFO バッファ |
| Mutex | 排他制御時に利用 |
| Semaphore | リソースの有無を管理 |
| Stream Buffer<br>Message Buffer | 1:1 で一方向の可変長データ通信 |

# Task Notification (タスク通知)

## タスクへ直接通知を行う

- 各タスクは通知状態と通知データ(32bit)を配列で管理
  - 通知や待ちを実行する際には、配列Indexを指定する
- 通知の際に、タスクが保持する通知データを特定の方法で変更可能
  - *eNoAction / eSetBits / eIncrement / eSetValueWithOverwrite / eSetValueWithoutOverwrite*
- 他のタスク間通信と比べオーバーヘッドが少ない

- 主要API
  - xTaskNotifyWait() / xTaskNotifyWaitIndexed()
  - xTaskNotify() / xTaskNotifyFromISR() / xTaskNotifyIndexed()

https://freertos.org/RTOS-task-notifications.html

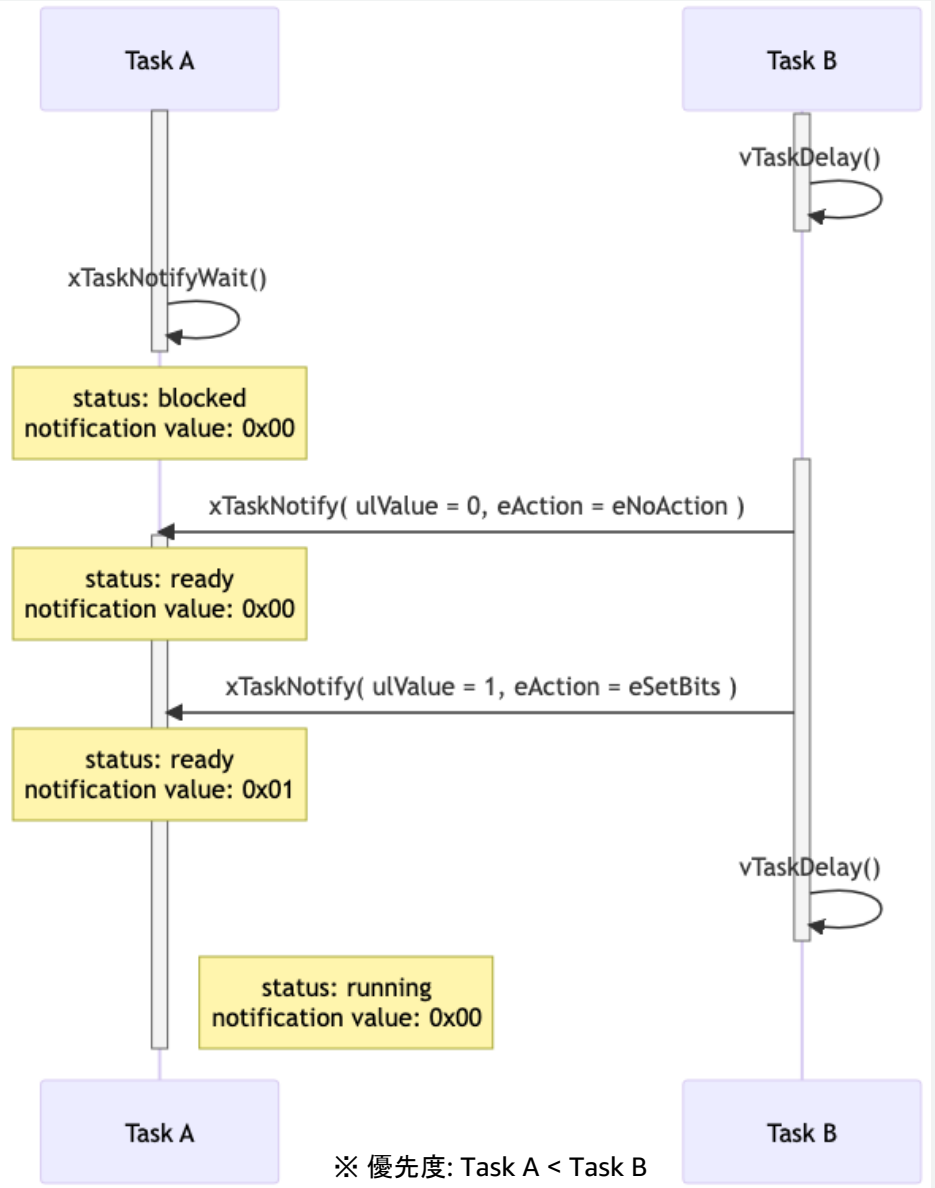
# Task Notification の特徴

Task A

```
uint32_t ulNotifiedValue;
xTaskNotifyWait(
     /* Specify bits to clear on entry. */
     0xFFFFFFFF,
     /* Specify bits to clear on exit. */
     0x01,
     /* Notified value. */
     &ulNotifiedValue,
     /* Block indefinitely. */
     portMAX_DELAY
);
```

Task B

```
xTaskNotify( xTaskB, 0, eNoAction );

xTaskNotify( xTaskB, 1, eSetBits );
```



# Queue

## 固定長データを扱うスレッドセーフなFIFOバッファ

- Queueの操作が出来ない場合は指定時間ブロックされる※
  - Empty時のReceive, Full時のSend, 別タスクが操作中など

## 主要API

- xQueueCreate()
- xQueueSendToBack() / xQueueSendToBackFromISR()
- xQueueReceive() / xQueueReceiveFromISR()

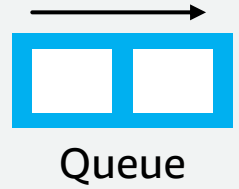
※ ISR内での実行はブロックされずエラーとなります。

https://www.freertos.org/a00018.html

# Queue の特徴

タスク起動前

```
QueueHandle_t xQueue;
void main( void ) {
    xQueue = xQueueCreate(2, sizeof( unsigned long ) );
    ...
}
```




TaskB

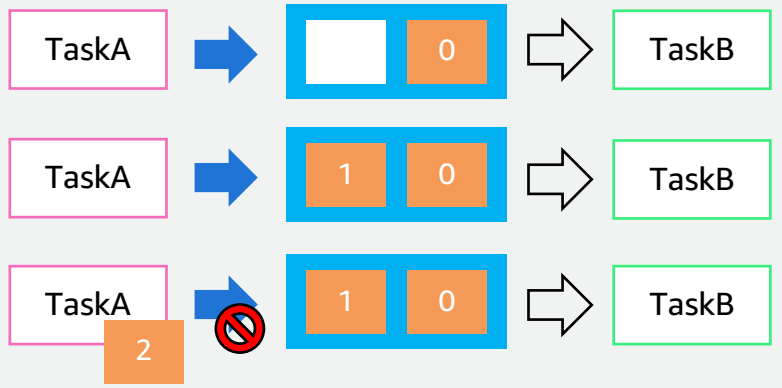
```
xQueueReceive( xQueue, ( void* ) &ulVarB, portMAX_DELAY );
```


TaskA

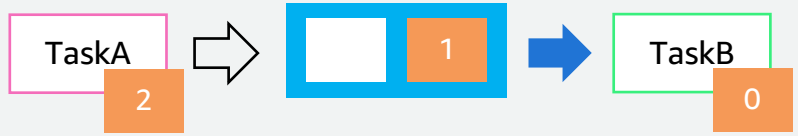
```
ulVarA = 0;
xQueueSendToBack( xQueue, ( void* ) &ulVarA, portMAX_DELAY );
```

```
ulVarA = 1;
xQueueSendToBack( xQueue, ( void* ) &ulVarA, portMAX_DELAY );
```

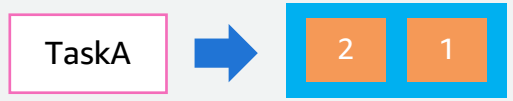
```
ulVarA = 2;
xQueueSendToBack( xQueue, ( void* ) &ulVarA, portMAX_DELAY );
```




TaskB




TaskA




※ 優先度: Task A > Task B

# Mutex

主に複数タスクでアクセスするリソースの排他制御に利用

- 優先度継承 機能付き
- 名前に Recursive が入っているAPIは、再起的に呼び出し可能

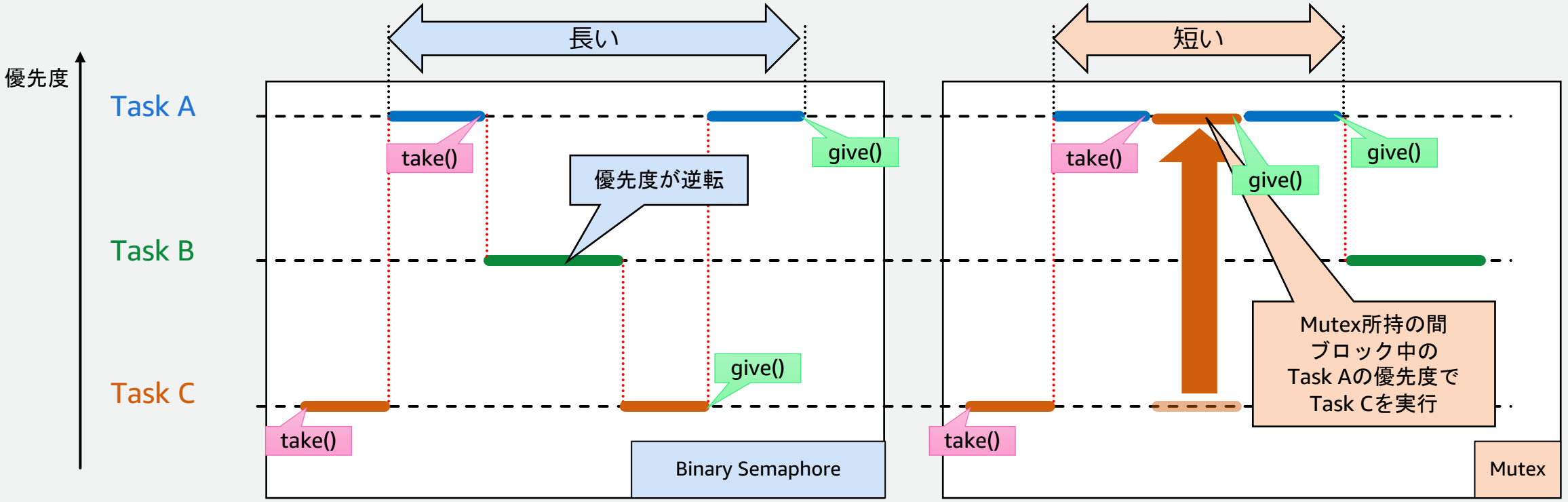
主要API

- xSemaphoreCreateMutex() / xSemaphoreCreateRecursiveMutex()
- xSemaphoreTake() / xSemaphoreTakeRecursive()
- xSemaphoreGive() / xSemaphoreGiveRecursive()

https://www.freertos.org/a00113.html

# Mutex の優先度継承

Mutexを取得しようとしてブロックされたタスクの優先度が、
先に取得していたタスクより高い場合、Mutexを所持するタスクが
ブロックされたタスクの優先度を一時的に継承する



# Semaphore (セマフォ) について

主にリソースの有無を管理

- 中身はQueueで実現 → 各要素のデータサイズ(uxItemSize)を0で作成したものと同等
- Task Notification や Mutex の方が、機能や性能面で適切な場合が多い (高速 & 省メモリ)

Counting Semaphore: https://freertos.org/RTOS_Task_Notification_As_Counting_Semaphore.html

- イベント数やリソース管理などに利用
- 1:1で一方向の利用であれば、Task Notificationで代替する方が性能が良い
  ( i.e.: ulTaskNotifyTakeIndexed / xTaskNotifyGiveIndexed )

Binary Semaphore: https://freertos.org/RTOS_Task_Notification_As_Binary_Semaphore.html

- 主に排他制御やタスク間同期に利用
- ただし優先度継承機能が無いので、排他制御には Mutex 利用を推奨
- 1:1で一方向のタスク間同期であれば、Task Notificationで代替する方が性能が良い
  ( i.e.: ulTaskNotifyIndexed / xTaskNotifyWaitIndexed )

# Stream Buffer / Message Buffer

## タスク間で可変長データの送受信に利用

- バッファが Full、Empty、データ不足等で操作できない場合は指定時間ブロックされる
- データの送信側と受信側が1:1の一方向通信が前提
  - Send / Receive のAPIは非スレッドセーフ
  - 内部ではTask Notification (uxIndexToNotify:0) を使って、Taskのブロックを解除

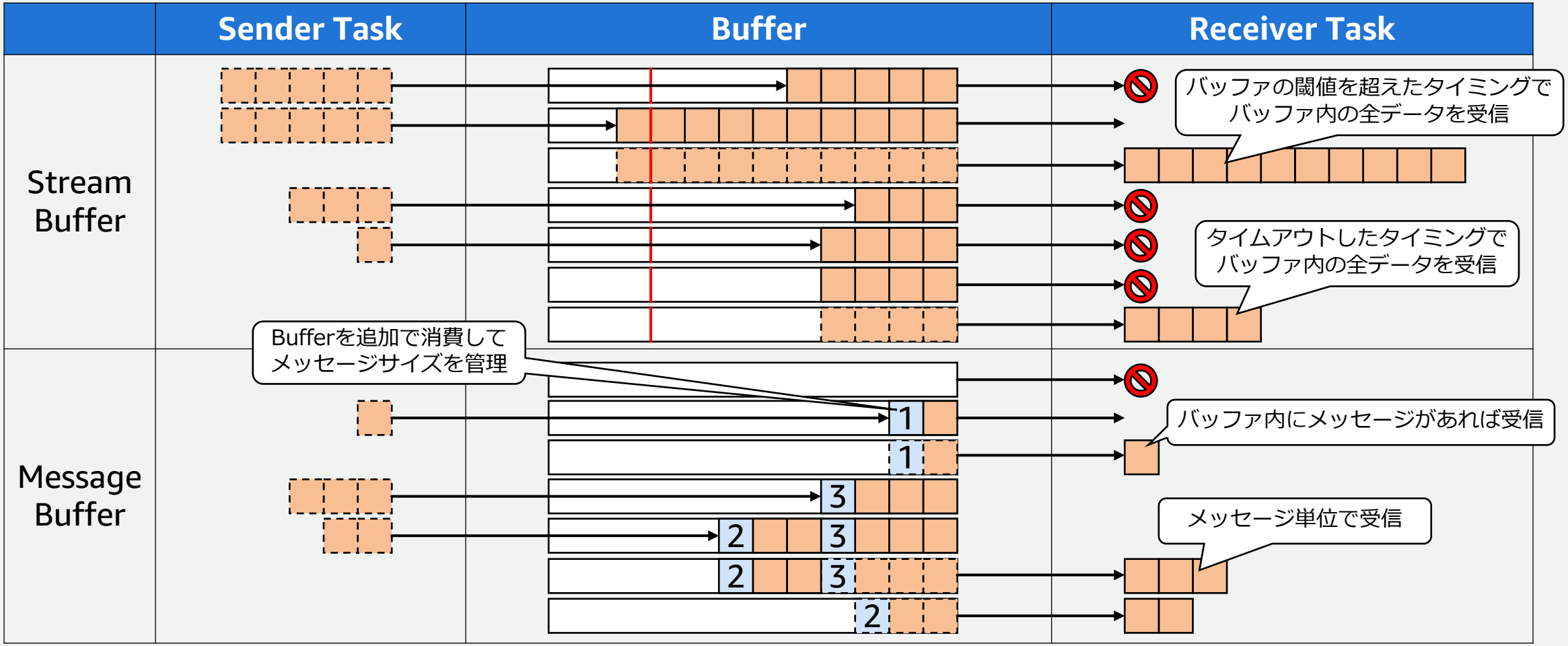
## Stream Buffer (バイトストリーム向け)

- バッファ内のデータ量が閾値を超えてから、受信側へデータを渡す

## Message Buffer (可変長データブロック向け)

- バッファ内で送信毎のデータサイズが管理され、受信側へその単位でデータを渡す

# Stream Buffer と Message Buffer の違い



# ソフトウェアタイマー

# ソフトウェアタイマー

## 指定したタイミングでコールバック

- Timer Service Taskから実行される
  - コールバック内でのブロッキングは想定されていない
- 2種類のタイマーを提供

## One-Shot

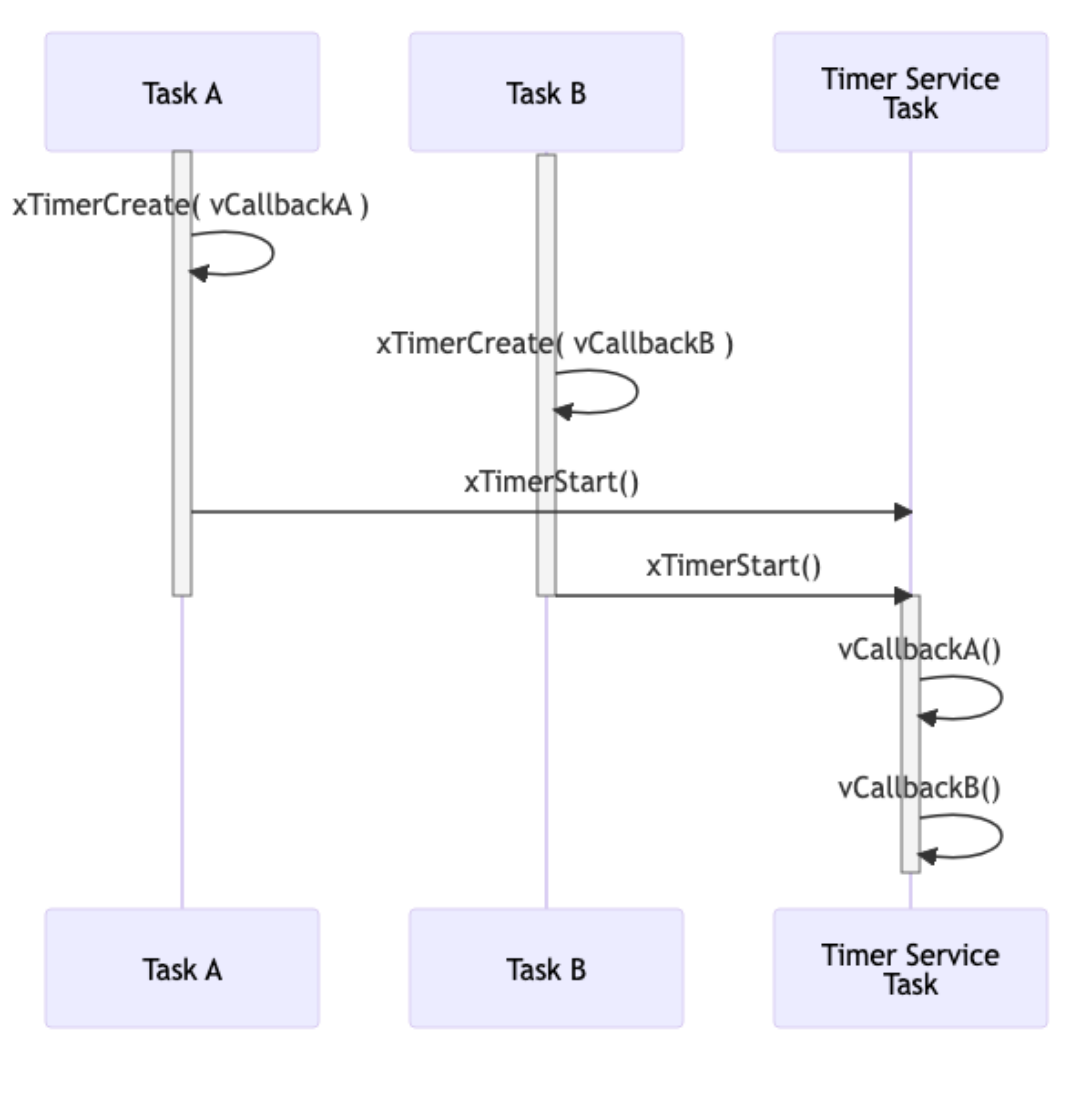
- 指定タイミングで1度だけコールバック

## Auto-Reload

- 指定間隔で繰り返しコールバック

## 主要API

- xTimerCreate()
- xTimerStart()



# Kernel コンフィグレーション

# Kernel コンフィグレーション

## FreeRTOSConfig.h 内でKernelの各種機能を調整可能

| マクロ名 (default値) | 内容 |
|---|---|
| configTICK_RATE_HZ (-) | Tick割り込み頻度 [Hz] ※ |
| configUSE_TIME_SLICING (1) | 同じ優先度内で時分割スケジューリングを行う※<br>(0: 待ち行列方式、1: 時分割方式) |
| configUSE_TASK_NOTIFICATIONS (1)<br>configTASK_NOTIFICATION_ARRAY_ENTRIES (1) | タスク通知機能の有無(0:無、1:有)や、通知配列の数 |
| configUSE_MUTEXES (0)<br>configUSE_RECURSIVE_MUTEXES (0) | Mutexの有無(0:無、1:有)や、オプション機能設定 |
| configMESSAGE_BUFFER_LENGTH_TYPE | Message Buffer内の各ブロックヘッダの型 |
| configUSE_TIMERS (0)<br>configTIMER_TASK_PRIORITY (-) | ソフトウェアタイマーの有無(0:無、1:有)や、機能設定 |
| configUSE_COUNTING_SEMAPHORES (0) | Counting  Semaphoreの有無(0:無、1:有) |

※設定変更に関してはチップベンダ、ハードウェアベンダへご確認下さい。

https://freertos.org/a00110.html

# アジェンダ

1. はじめに
2. FreeRTOSとは
3. FreeRTOS Kernel
4. **FreeRTOS ライブラリ概要**
5. はじめかた
6. まとめ

# FreeRTOS ライブラリの特徴

**Local Connectivity**

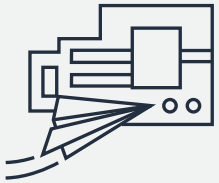
クラウド接続無しで
AWS IoT Greengrass
デバイスと通信

**Cloud Connectivity**

マイコンデバイスから
簡単にデータを収集、
アクションを実行

**Security**

デバイスのデータと
接続をセキュアに



**OTA and Code Signing**

セキュリティ更新や
バグ修正などの
ファームウェアをデプロイ

# 汎用機能ライブラリ (一部抜粋)

| ライブラリ名 | 説明 |
| --- | --- |
| FreeRTOS-Plus-TCP | FreeRTOS向け TCP/IP スタック (UDP含む) |
| coreMQTT | MQTT クライアント<br>AWS IoT Coreへの接続も可能 |
| coreHTTP | HTTP/1.1 クライアント |
| coreJSON | JSON パーサー |
| coreSNTP | SNTPv4 クライアント |
| Cellular Interface | 汎用的なATコマンドをAPIとして提供 |

https://freertos.org/FreeRTOS-Plus/index.html
https://freertos.org/freertos-core/overview.html

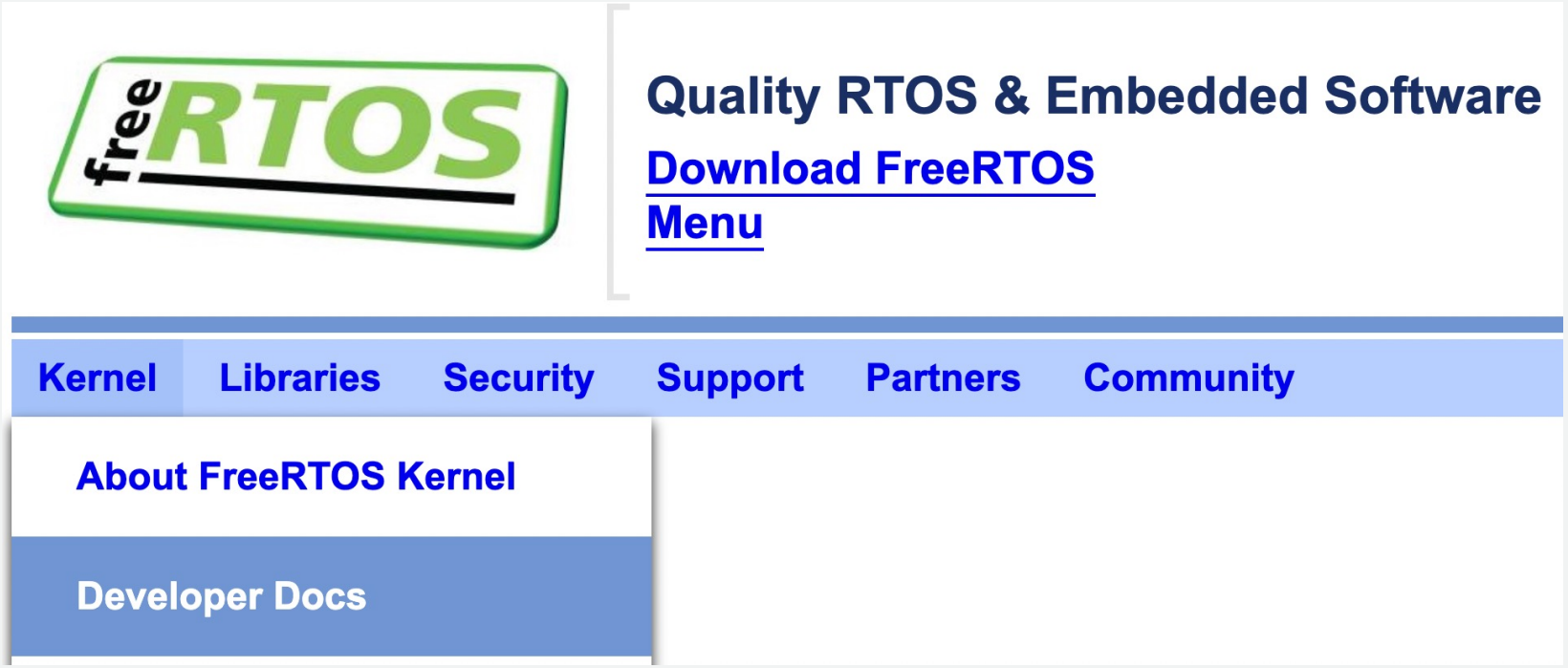
# AWS 関連ライブラリ

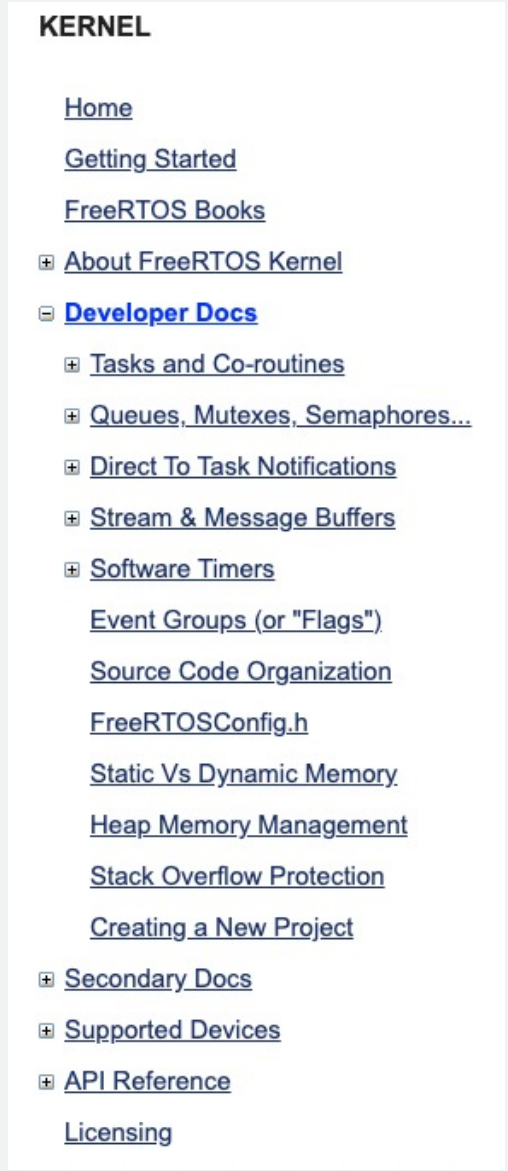
| ライブラリ名 | 説明 |
| --- | --- |
| AWS SigV4 | AWS Signature Version4 署名機能の提供 |
| AWS IoT OTA | OTA 更新ジョブ実行に必要な一連の処理とインターフェースを提供 |
| AWS IoT Device Shadow | 各サービス利用に必要なTopic名の生成やイベント処理に関する機能の提供 |
| AWS IoT Device Defender | |
| AWS IoT Jobs | |
| AWS IoT Fleet Provisioning | |

https://freertos.org/iot-libraries.html

# FreeRTOS のドキュメント

- FreeRTOS.org に情報を集約
- API リファレンスや、各機能の詳細説明を記載



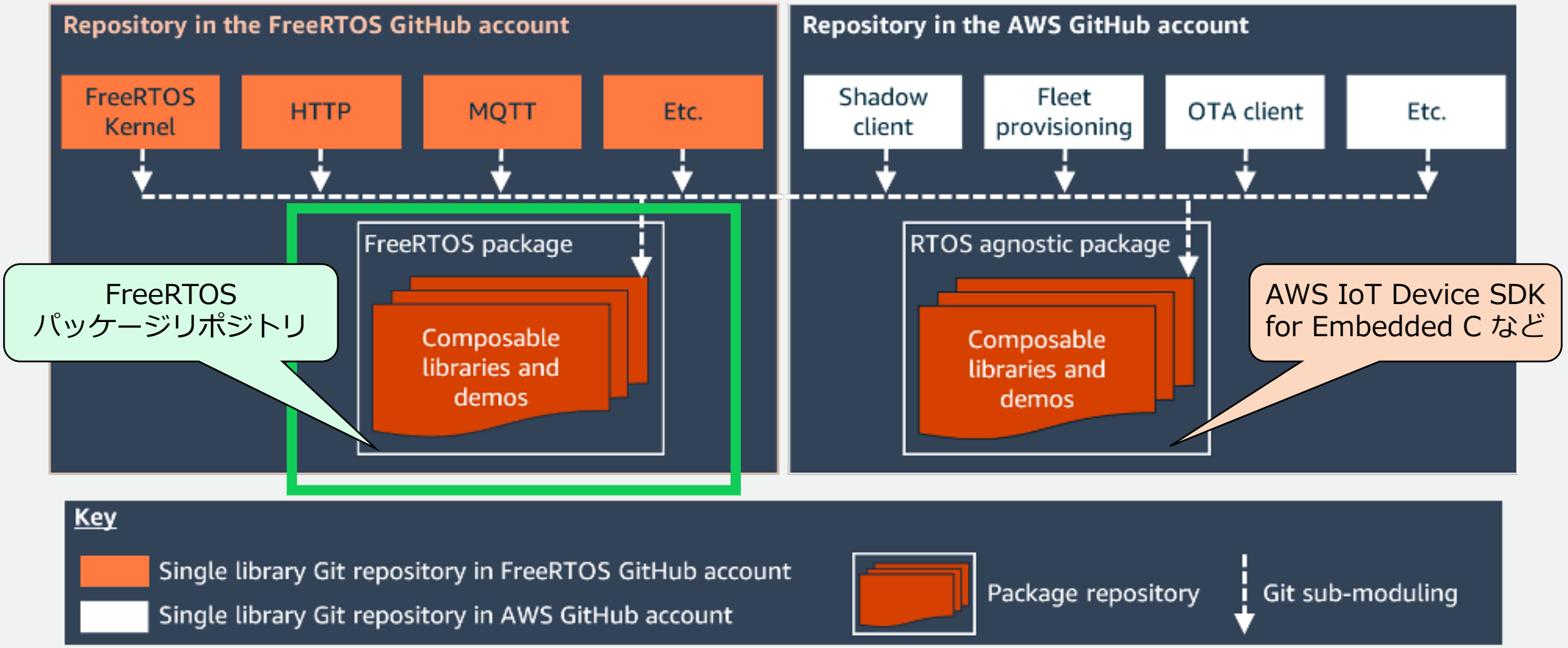




https://freertos.org/features.html

# アジェンダ

1. はじめに
2. FreeRTOSとは
3. FreeRTOS Kernel
4. FreeRTOS ライブラリ概要
5. **はじめかた**
6. まとめ

# リポジトリ構成



# FreeRTOS パッケージリポジトリ

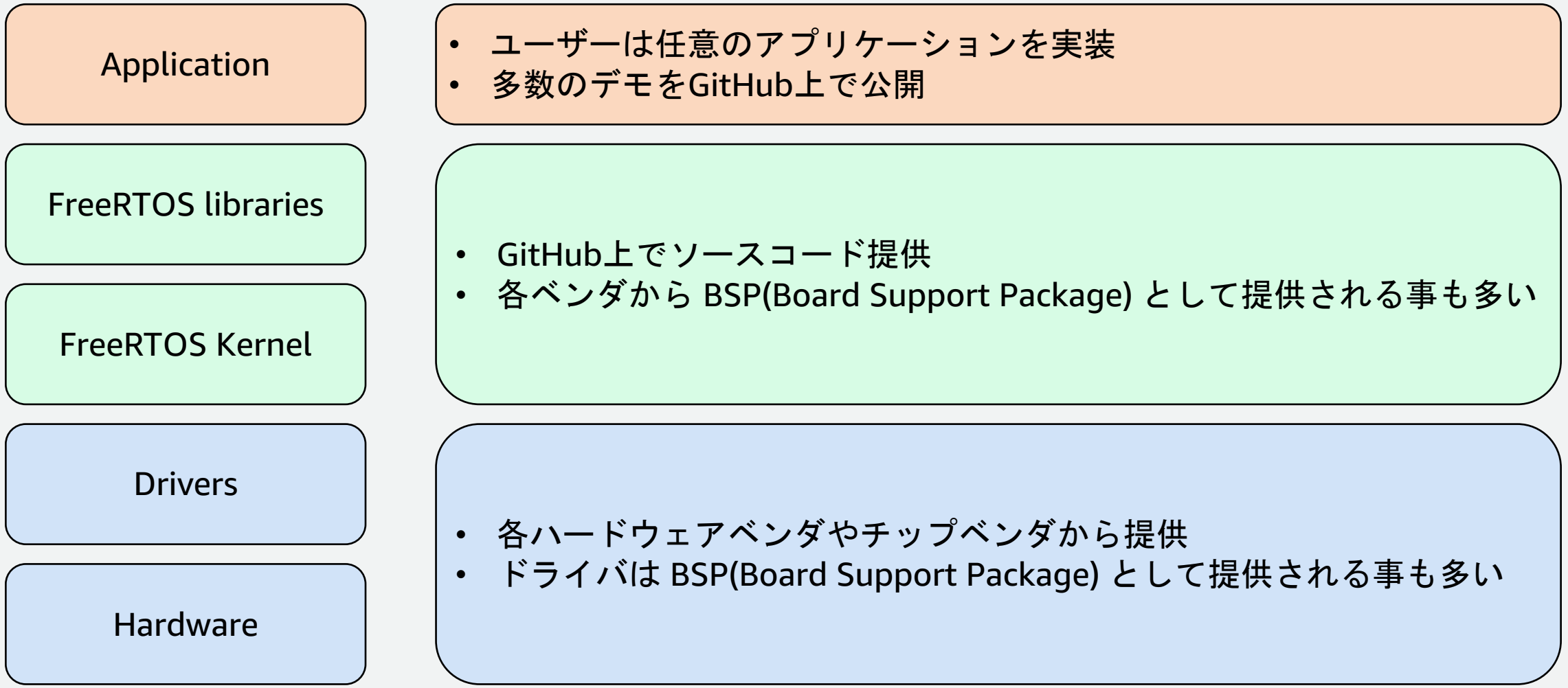
## FreeRTOS

- https://github.com/FreeRTOS/FreeRTOS
- デモを含めた統合リポジトリ

## FreeRTOS-LTS

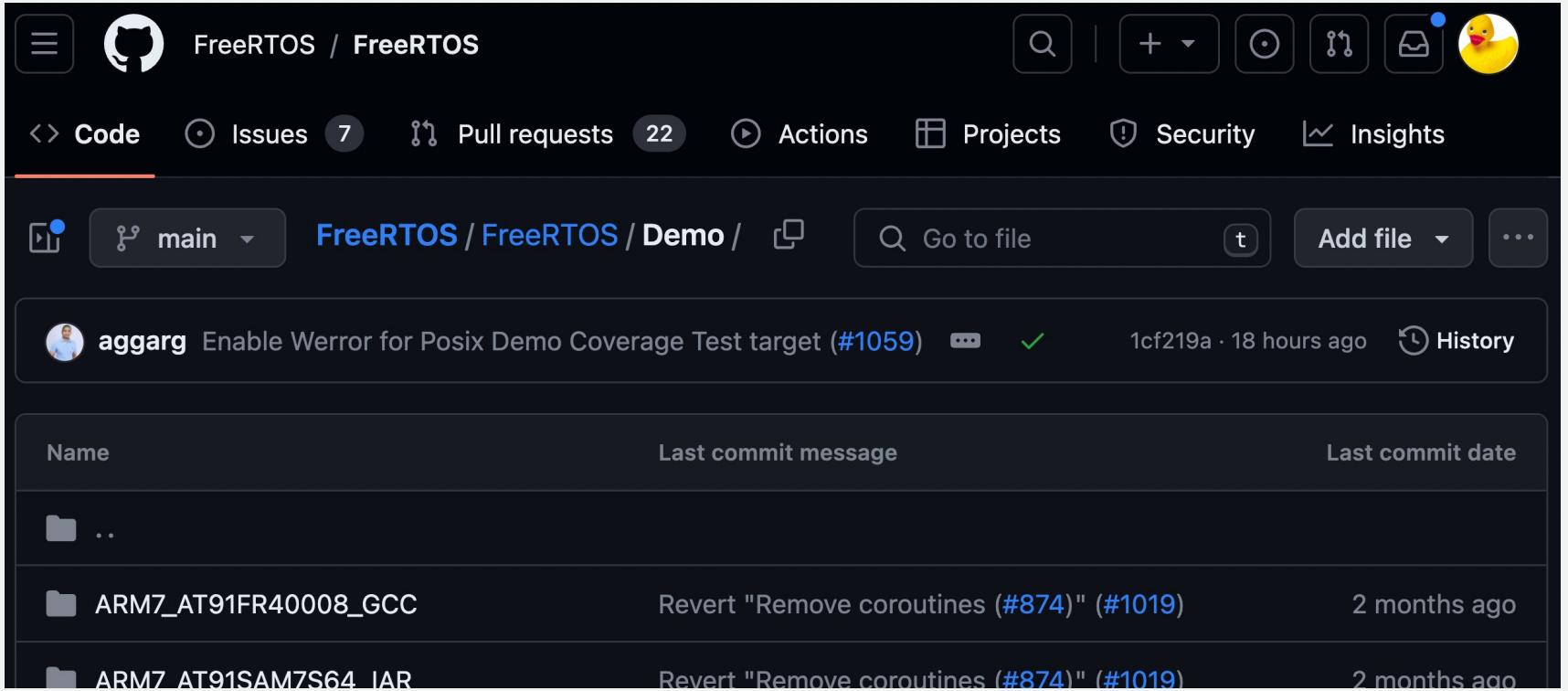
- https://github.com/FreeRTOS/FreeRTOS-LTS
- LTS対象のライブラリ定義しパッケージしたリポジトリ

# FreeRTOS 実行環境の構成



# FreeRTOS のデモアプリケーション

- リポジトリ内には各チップ、ボード向けのサンプルデモを同梱
- 各ベンダからも別途サンプルデモを多数提供




https://github.com/FreeRTOS/FreeRTOS/tree/main/FreeRTOS/Demo

# FreeRTOS を試してみる (QEMU上で擬似的に実行) 1/3

## 1. Ubuntu環境でARM Cortex-M3 のQEMU環境をインストール

```
# Install QEMU & ARM architectures
$ sudo apt-get install -y make qemu qemu-system-arm
```

## 2. ARMのビルド環境を構築

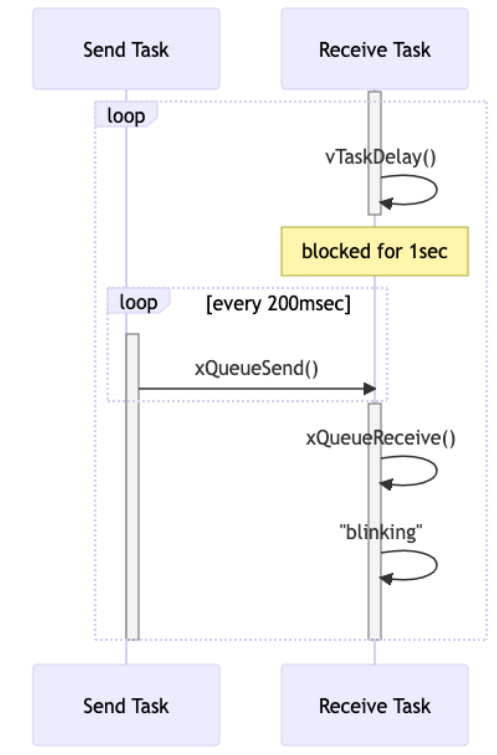
```
# Download arm-none-eabi tools
$ curl https://armkeil.blob.core.windows.net/developer/Files/downloads/gnu-rm/10-2020q4/gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2 -o gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2
$ tar -xjvf gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2

$ export PATH="$PWD/gcc-arm-none-eabi-10-2020-q4-major/bin:$PATH"
```

https://github.com/FreeRTOS/FreeRTOS/blob/main/FreeRTOS/Demo/CORTEX_M3_MPS2_QEMU_GCC/Readme.md

# FreeRTOS を試してみる (QEMU上で擬似的に実行) 2/3

## 3. デモを含んだ FreeRTOS コードをリポジトリから取得

```
# Clone FreeRTOS & Demo from a repository
$ git clone https://github.com/FreeRTOS/FreeRTOS.git -b 202212.01
$ cd FreeRTOS
$ git submodule update --init --depth 1
$ cd ./FreeRTOS/Demo/CORTEX_M3_MPS2_QEMU_GCC
```




```
static void prvQueueSendTask( void * pvParameters )
{
	TickType_t xNextWakeTime;
	const uint32_t ulValueToSend = 100UL;

	xNextWakeTime = xTaskGetTickCount();

	for( ; ; )
	{
		vTaskDelayUntil( &xNextWakeTime,
		mainQUEUE_SEND_FREQUENCY_MS );

		xQueueSend( xQueue, &ulValueToSend, 0U );
	}
}
```

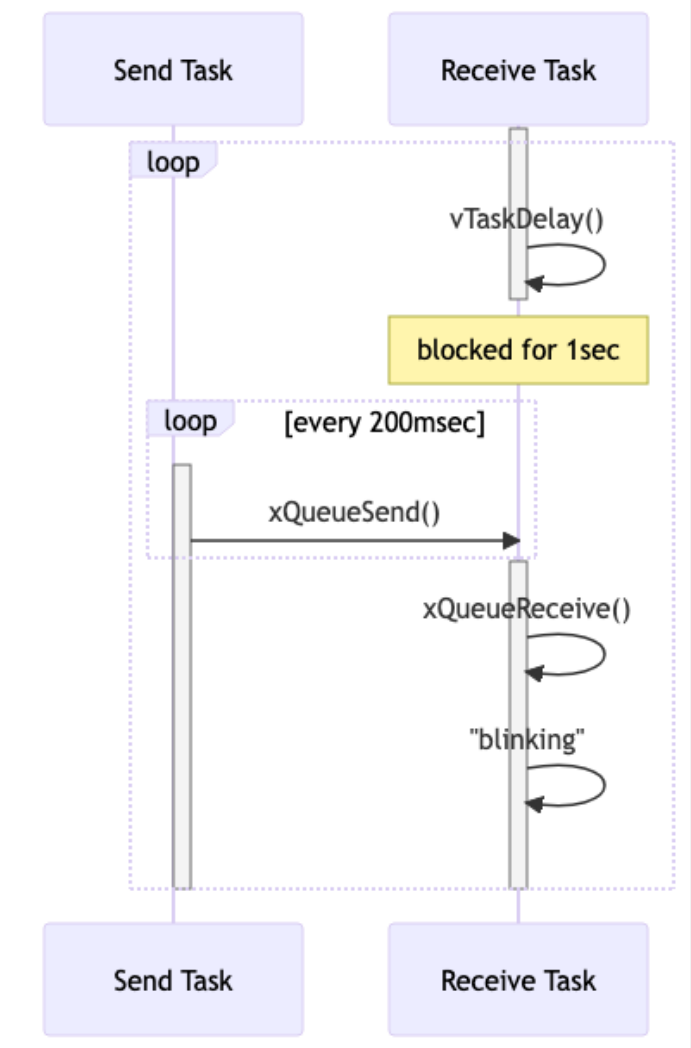
```
static void prvQueueReceiveTask( void * pvParameters )
{
	uint32_t ulReceivedValue;
	const uint32_t ulExpectedValue = 100UL;

	for( ; ; )
	{
		xQueueReceive( xQueue, &ulReceivedValue,
		portMAX_DELAY );

		if( ulReceivedValue == ulExpectedValue )
		{
			printf( "%s¥n", "blinking" );
			vTaskDelay( 1000 );
			ulReceivedValue = 0U;
			ulRxEvents++;
		}
	}
}
```

https://github.com/FreeRTOS/FreeRTOS/blob/main/FreeRTOS/Demo/CORTEX_M3_MPS2_QEMU_GCC/Readme.md
https://github.com/FreeRTOS/FreeRTOS/blob/main/FreeRTOS/Demo/CORTEX_M3_MPS2_QEMU_GCC/main_blinky.c

# FreeRTOS を試してみる (QEMU上で擬似的に実行) 3/3

## 4. デモのビルド & 実行

```
# Run the Blinky Demo on ARM Cortex-M3 in Qemu
$ make
$ sudo qemu-system-arm -machine mps2-an385 -monitor null ¥
-semihosting --semihosting-config enable=on,target=native ¥
-kernel ./build/RTOSDemo.axf -serial stdio -nographic

blinking
blinking
blinking
blinking
blinking
blinking
...
```

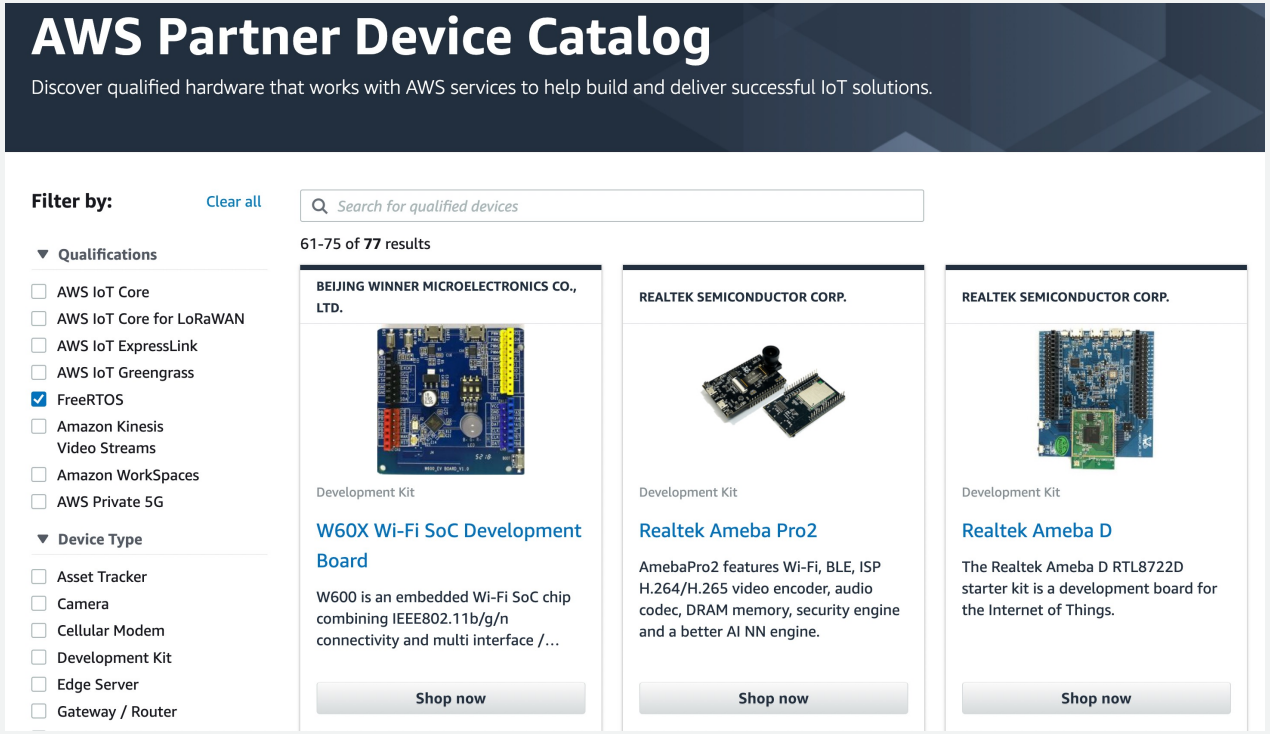



https://github.com/FreeRTOS/FreeRTOS/blob/main/FreeRTOS/Demo/CORTEX_M3_MPS2_QEMU_GCC/Readme.md

# AWS Partner Device Catalog
(FreeRTOS対応デバイスを見つける)

FreeRTOSに対応する製品の一覧

- 各ベンダのチップやモジュール、評価ボードが登録されている
- ドキュメントや購入先のリンクを掲載




https://devices.amazonaws.com/search?sv=freertos

# アジェンダ

1. はじめに
2. FreeRTOSとは
3. FreeRTOS Kernel
4. FreeRTOS ライブラリ概要
5. はじめかた
6. **まとめ**

# まとめ

- FreeRTOS は、MPU/MCU上で複数タスクを並行して実行できる
  - 優先度によるタスク設計を行うことで、リアルタイム処理を実現
  - 処理のモジュール化によって、開発/保守の効率を向上
- FreeRTOS ライブラリによってIoTの機能を実現
  - 価値を生み出すエッジ側やクラウド側の処理に注力
- ドキュメントやデモといった豊富なリファレンス
  - IoT デバイス開発の工数を削減

# 参照情報

## FreeRTOS

- Offical site: https://freertos.org/
- About LTS: https://www.freertos.org/lts-libraries.html
- About EMP: https://www.freertos.org/EMP.html
- AWS Document: https://docs.aws.amazon.com/freertos/
- リポジトリ
  - FreeROS: https://github.com/FreeRTOS/FreeRTOS
  - LTS: https://github.com/FreeRTOS/FreeRTOS-LTS

## AWS Partner Device Catalog

- https://devices.amazonaws.com/search?sv=freertos

# AWS Black Belt Online Seminar とは

- 「サービス別」「ソリューション別」「業種別」などのテーマに分け、アマゾン ウェブ サービス ジャパン合同会社が提供するオンラインセミナーシリーズです
- AWS の技術担当者が、AWS の各サービスやソリューションについてテーマごとに動画を公開します
- 以下の URL より、過去のセミナー含めた資料などをダウンロードすることができます
  - https://aws.amazon.com/jp/aws-jp-introduction/aws-jp-webinar-service-cut/
  - https://www.youtube.com/playlist?list=PLzWGOASvSx6FIwIC2X1nObr1KcMCBBlqY

ご感想は X (Twitter) へ！ハッシュタグは以下をご利用ください
#awsblackbelt

# 内容についての注意点

- 本資料では資料作成時点のサービス内容および価格についてご説明しています。AWS のサービスは常にアップデートを続けているため、最新の情報は AWS 公式ウェブサイト (https://aws.amazon.com/) にてご確認ください
- 資料作成には十分注意しておりますが、資料内の価格と AWS 公式ウェブサイト記載の価格に相違があった場合、AWS 公式ウェブサイトの価格を優先とさせていただきます
- 価格は税抜表記となっています。日本居住者のお客様には別途消費税をご請求させていただきます
- 技術的な内容に関しましては、有料の AWS サポート窓口へお問い合わせください
- 料金面でのお問い合わせに関しましては、カスタマーサポート窓口へお問い合わせください (マネジメントコンソールへのログインが必要です)

# Thank you!