Université IBN TOFAIL – Kénitra
ECOLE NATIONALE DES SCIENCES APPLIQUEES

Cours Systèmes Temps Réel

CH4: Le Système d'Exploitation en Temps Réel: FreeRTOS

Par:

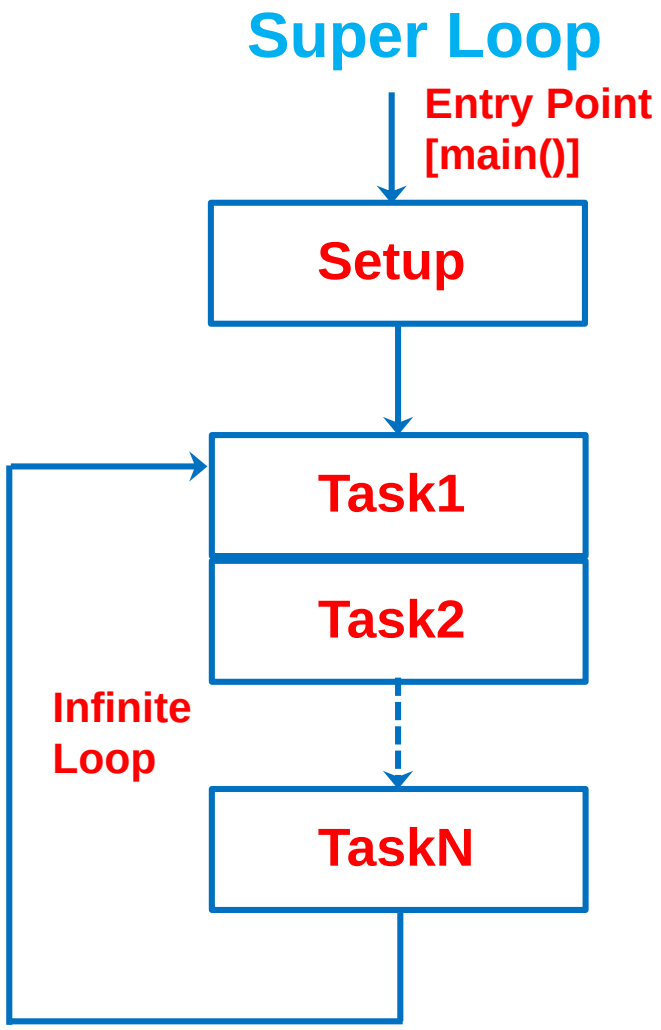
Hassan EL FADIL

elfadil.h@gmail.com

1. La Boucle Super (Super Loop) vs. RTOS

1.1. La Super Loop:

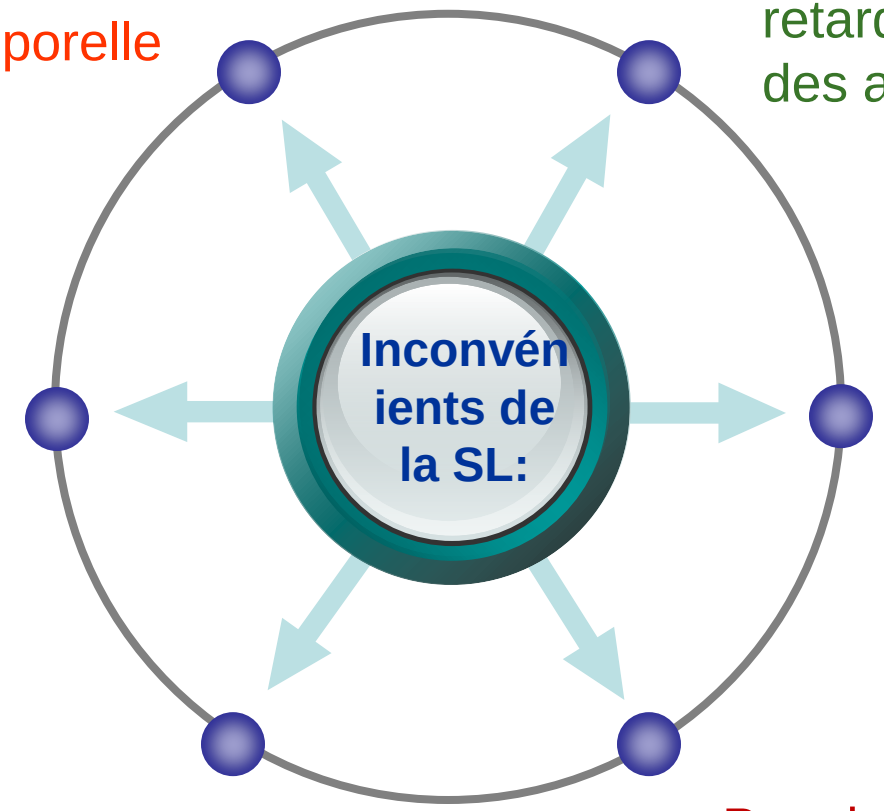
- L'architecture en **super boucle**, également connue sous le nom d'architecture en boucle principale, est une approche simple et directe pour gérer les tâches dans les systèmes basés sur un microcontrôleur.
- Dans cette architecture, la boucle principale du programme exécute de manière répétée une séquence de tâches, dont chacune prend un temps fixe.
- Les tâches sont planifiées séquentiellement et l'ordre d'exécution est généralement déterminé par l'ordre dans lequel elles apparaissent dans la boucle principale.
- Il n'y a pas de gestion de tâches ou de planificateur (ordonnanceur).



… La Super Loop vs. RTOS

Manque de Prédictibilité Temporelle

Les tâches critiques peuvent être retardées, car elles doivent attendre la fin des autres tâches dans la boucle



Difficile à gérer des applications complexes

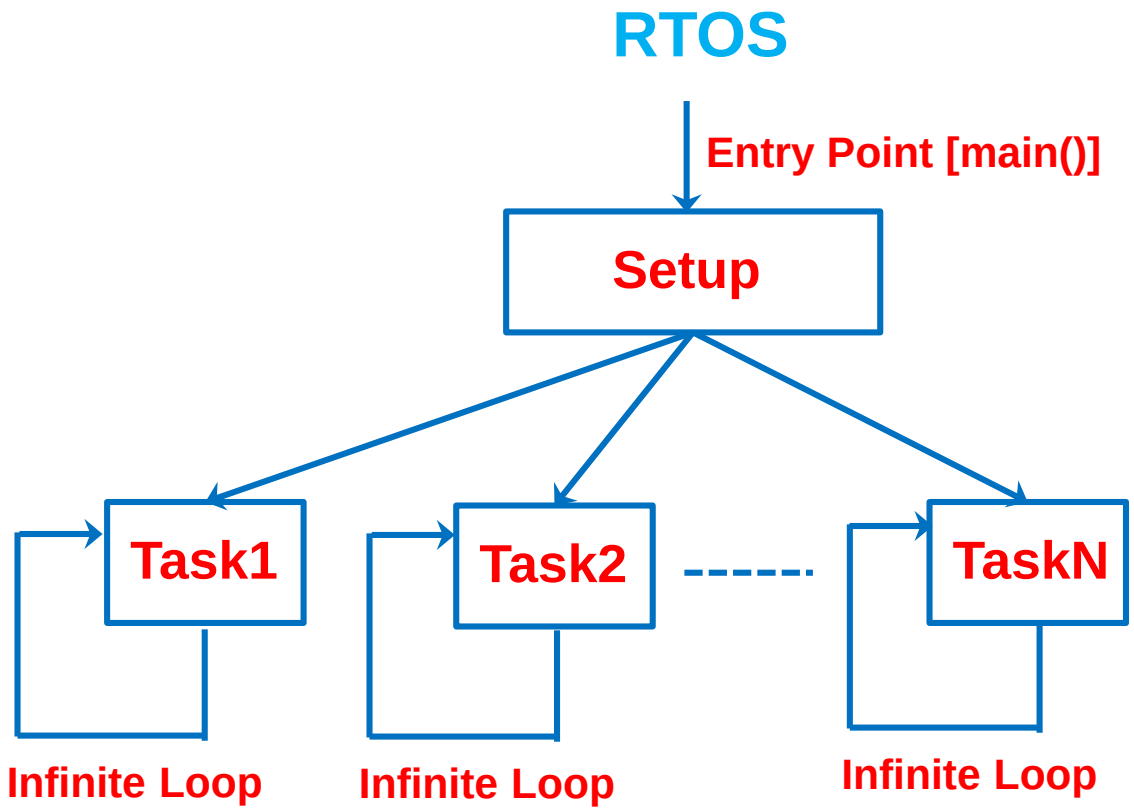
Scalabilité Limitée: À mesure que le système évolue, la boucle principale peut devenir trop complexe.

Absence de multitâche

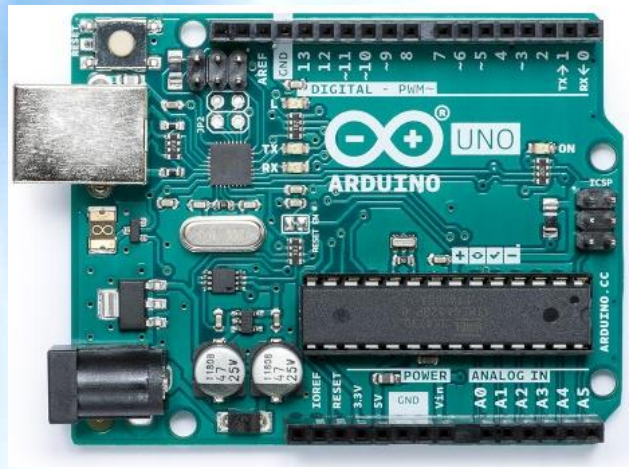
Pas de gestion de priorité

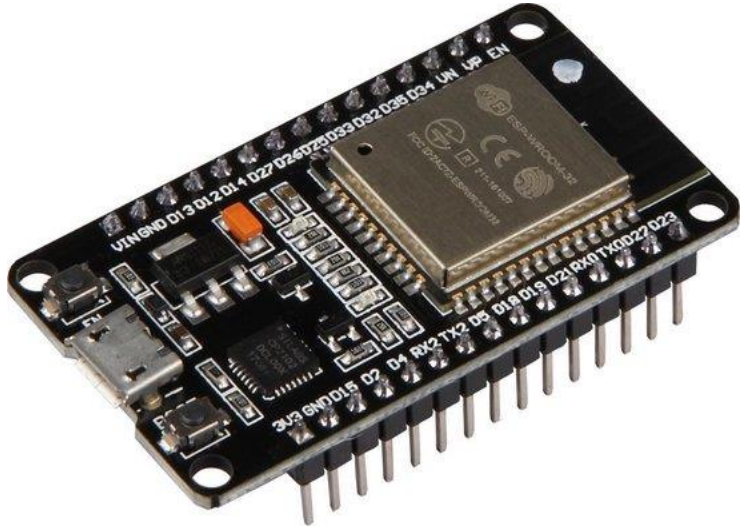
1.2. Le RTOS (comme FreeRTOS):

- Gestion des Tâches: les tâches sont gérées par un noyau, responsable de la planification, du changement de contexte et de l'allocation des ressources.
- Prédictibilité : Meilleure gestion des contraintes de temps réel.
- Complexité : Plus complexe à mettre en œuvre et à déboguer.
- Empreinte Mémoire : Nécessite plus de mémoire et de ressources.



… La Super Loop vs. RTOS





ATmega 328p
- 16 MHz
- 32 ko (flash)
- 2 ko RAM

STM32 F401RE
- 84 MHz
- 512 ko (flash)
- 96 ko RAM

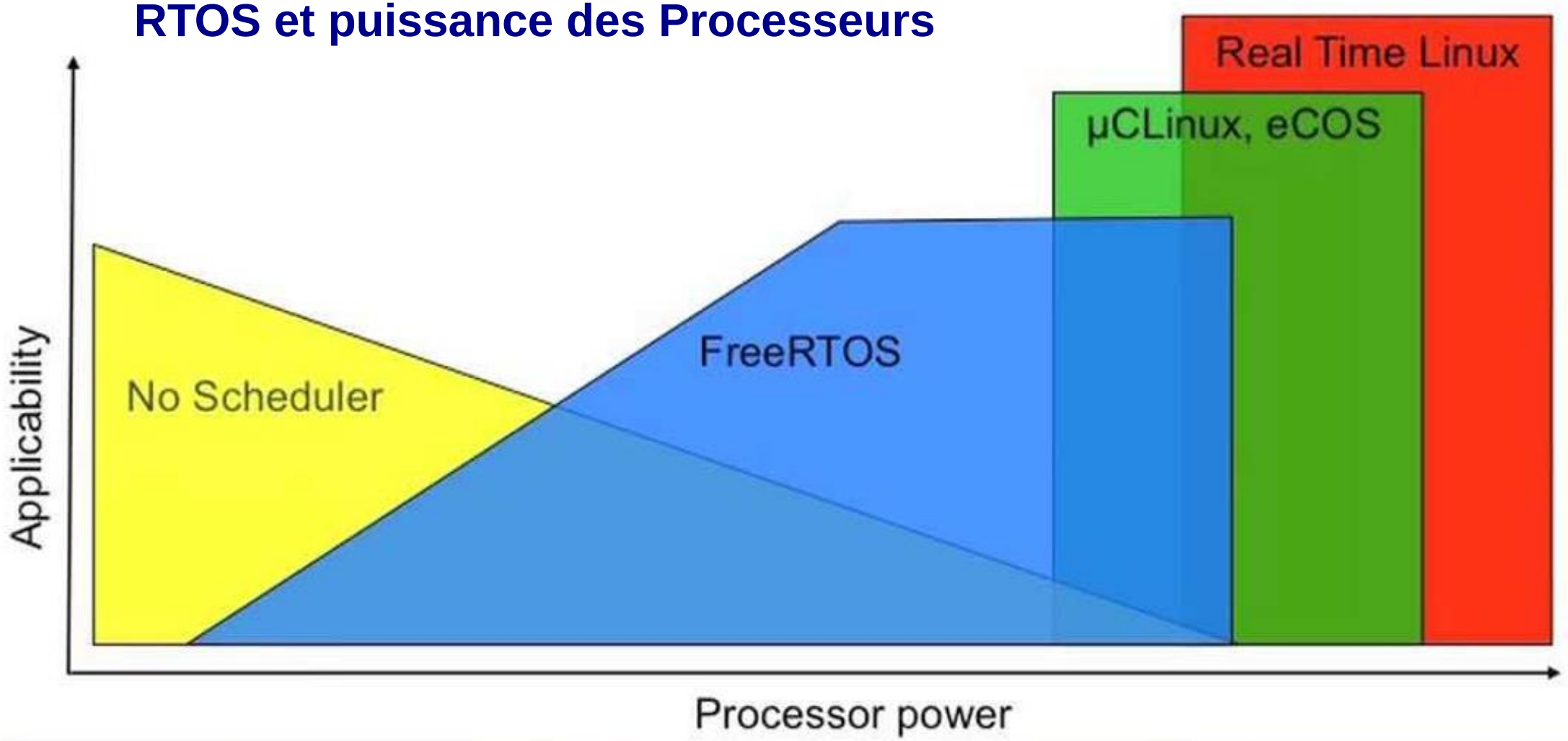
ESP-WROOM-32
- 240 MHz (dual core)
- 4 Mo (flash)
- 520 ko RAM

Super Loop → ***RTOS***

2. Les RTOS les plus populaires (2024):

- **Deos** (DDC-I)
- **embOS** (SEGGER)
- **FreeRTOS** (Amazon)
- **Integrity** (Green Hills Software)
- **Keil RTX** (ARM)
- **LynxOS** (Lynx Software Technologies)
- **MQX** (Philips NXP / Freescale)
- **Nucleus** (Mentor Graphics)
- **QNX Neutrino** (BlackBerry)
- **PikeOS** (Sysgo)
- **SafeRTOS** (Wittenstein)
- **ThreadX** (Microsoft Express Logic)
- **µC/OS** (Micrium)
- **VxWorks** (Wind River)
- **Zephyr** (Linux Foundation)

RTOS et puissance des Processeurs

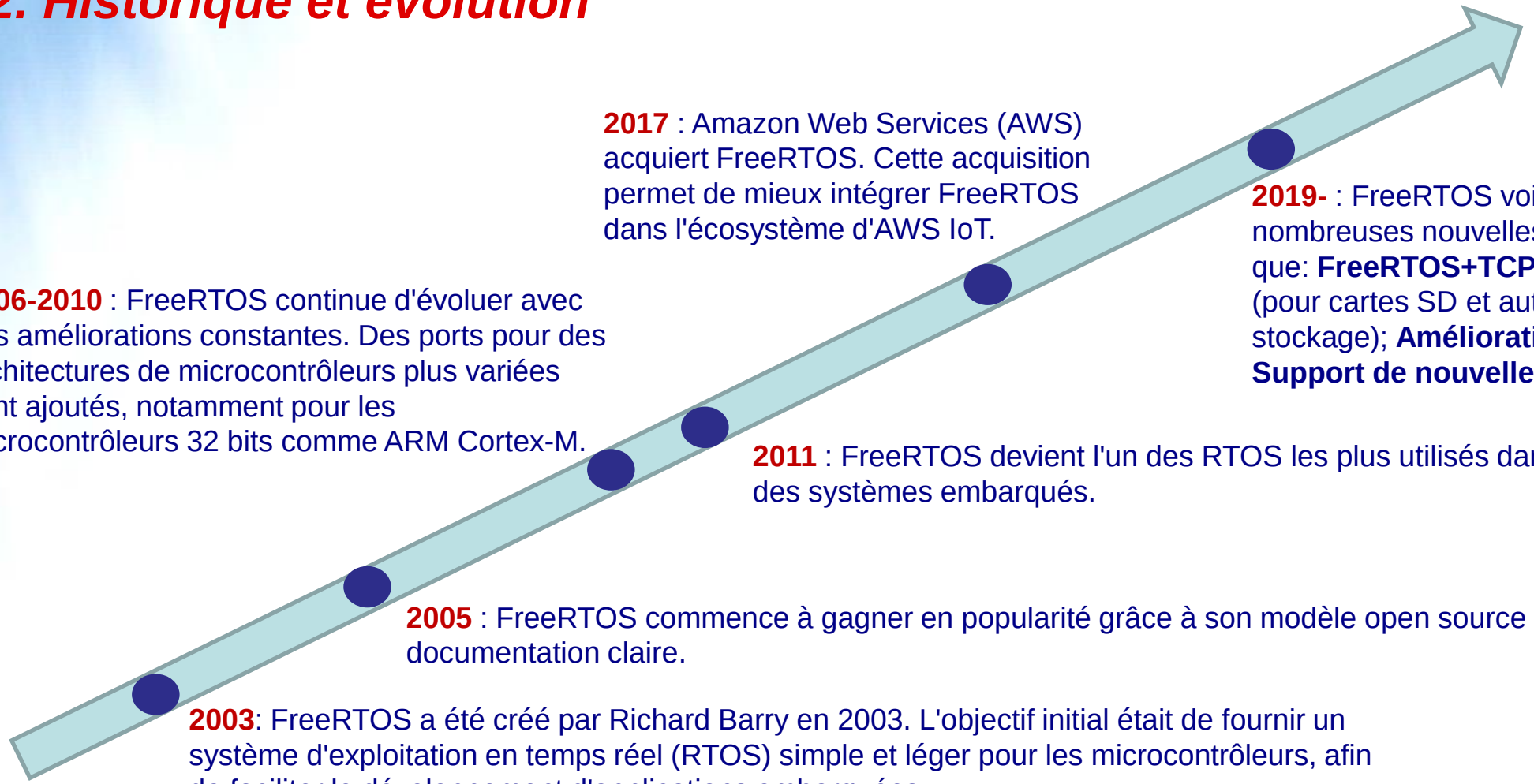


3. Présentation de FreeRTOS

3.1. Définition

- FreeRTOS: Free Real-Time Operating System
- Système d'exploitation en temps réel open source
- Conçu pour les microcontrôleurs et les petits microprocesseurs
- Il permet de gérer des tâches concurrentes de manière efficace, tout en respectant des contraintes de temps strictes.

3.2. Historique et évolution



2003: FreeRTOS a été créé par Richard Barry en 2003. L'objectif initial était de fournir un système d'exploitation en temps réel (RTOS) simple et léger pour les microcontrôleurs, afin de faciliter le développement d'applications embarquées

2005 : FreeRTOS commence à gagner en popularité grâce à son modèle open source et à sa documentation claire.

2006-2010 : FreeRTOS continue d'évoluer avec des améliorations constantes. Des ports pour des architectures de microcontrôleurs plus variées sont ajoutés, notamment pour les microcontrôleurs 32 bits comme ARM Cortex-M.

2011 : FreeRTOS devient l'un des RTOS les plus utilisés dans le domaine des systèmes embarqués.

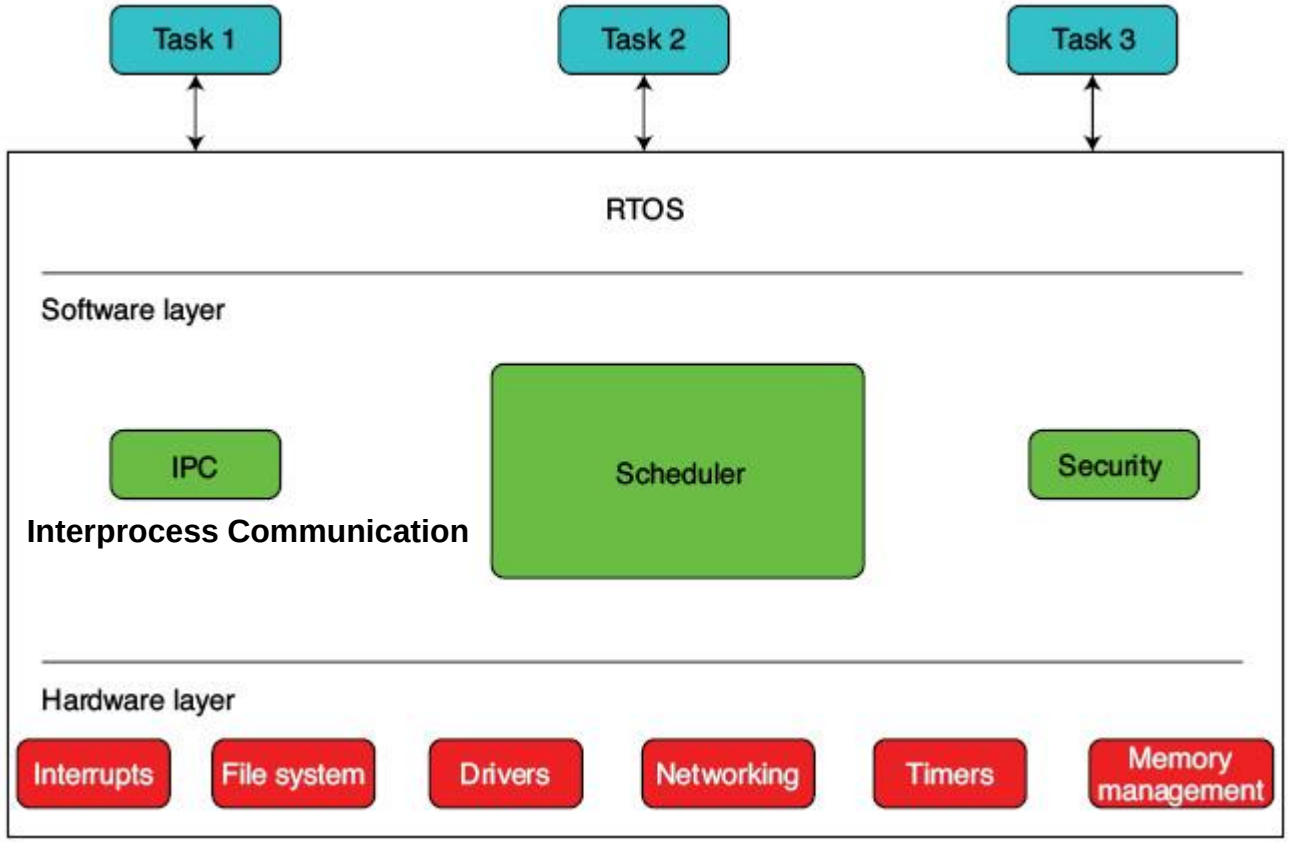
2017 : Amazon Web Services (AWS) acquiert FreeRTOS. Cette acquisition permet de mieux intégrer FreeRTOS dans l'écosystème d'AWS IoT.

2019- : FreeRTOS voit l'ajout de nombreuses nouvelles fonctionnalités, telles que: **FreeRTOS+TCP**; **FreeRTOS+FAT** (pour cartes SD et autres supports de stockage); **Améliorations de la sécurité**; **Support de nouvelles architectures**, etc.

4. Structure de Base de FreeRTOS

4.1. Introduction :

- FreeRTOS est un système d'exploitation temps réel (RTOS) open-source conçu pour les systèmes embarqués.
- Sa structure de base repose sur plusieurs composants clés qui permettent de gérer les tâches, les interruptions, la communication entre les tâches, les minuteries, etc.

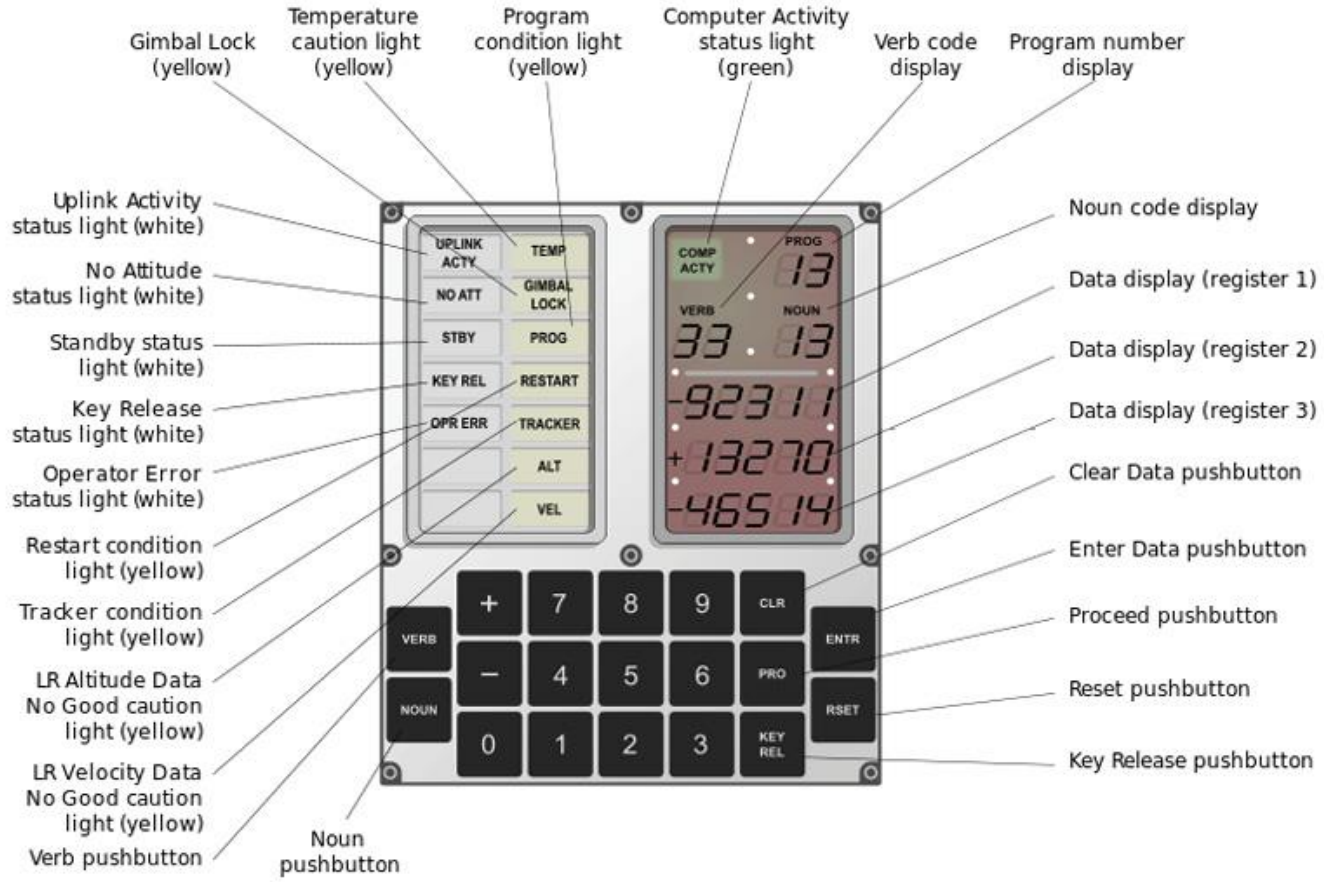


En **FreeRTOS**, les mécanismes de communication inter-tâches (**IPC: Interprocess Communication**) permettent aux tâches de partager des données et de se synchroniser efficacement dans un système temps réel. Voici un résumé des principaux outils :

- **Files d'attente (Queues)** : Permettent de transmettre des données entre tâches ou entre une interruption et une tâche. Fonctionnent en FIFO (First In, First Out).
- **Sémaphores** :
  - **Binaire** : Synchronisation simple entre tâches ou signaux d'interruption.
  - **Comptable** : Gestion de plusieurs ressources ou événements.
  - **Mutex** : Protection des ressources partagées avec priorité héritée pour éviter l'inversion de priorité.
- **Tampons (Buffers)** :
  - **Tampons de flux** : Transmission de flux de données (par exemple, pour la communication série).
  - **Tampons de messages** : Transmission de messages discrets de taille variable.
- Etc.

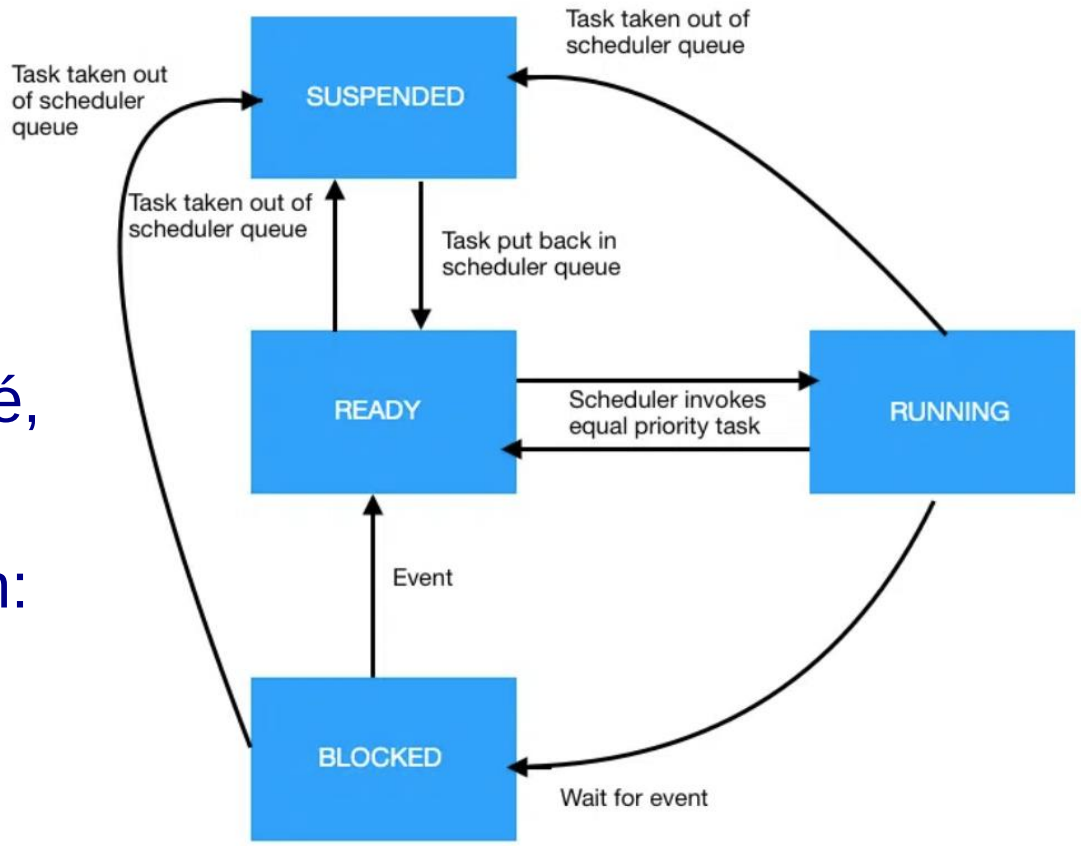
4.2. Les Tâches (Tasks) :

- Les tâches sont les unités de base de l'exécution en temps réel dans FreeRTOS.
- Chaque tâche est une fonction indépendante qui exécute du code de manière concurrente avec d'autres tâches.



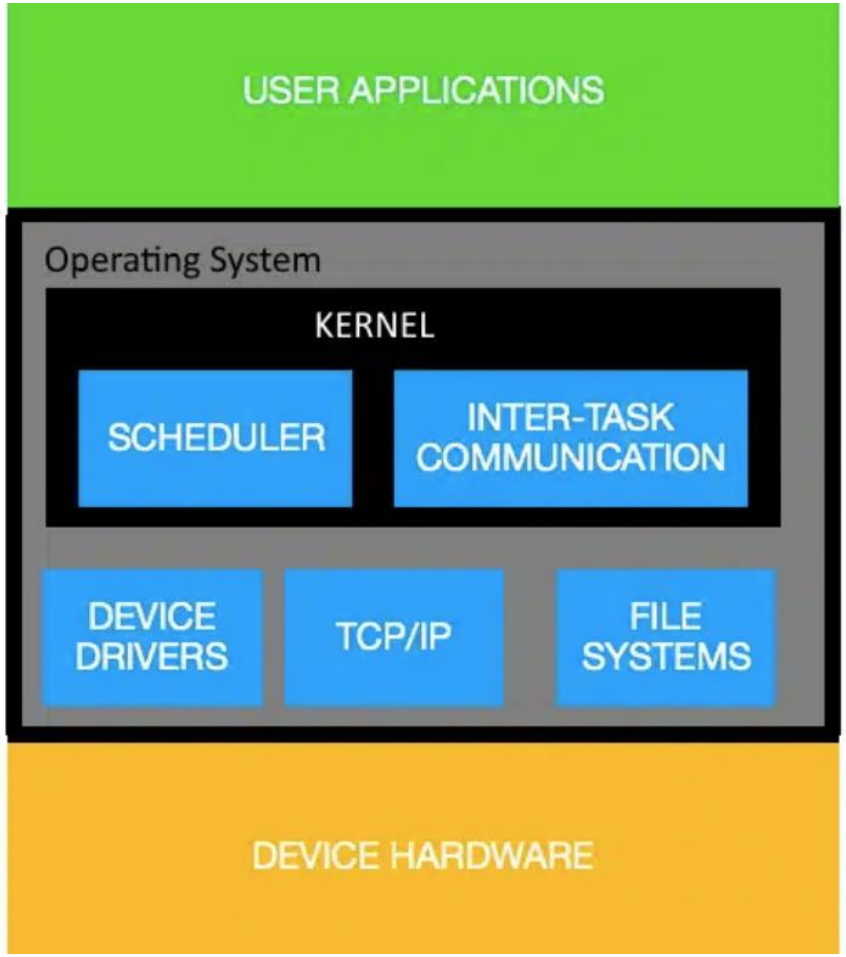
Exemple de tâches dans ne interface informatique de guidage du module lunaire (LM)

- Les tâches ont des priorités définies et FreeRTOS utilise un ordonnanceur pour déterminer quelle tâche doit s'exécuter à un moment donné.
- Une tâche peut passer par l'un des états suivants: En cours d'exécution, Prêt, Bloqué, Suspendu
- Les tâches sont créées à l'aide de la fonction:
  - **xTaskCreate() sous Arduino IDE**
  - **osThreadNew () sous STM32CubeIDE**



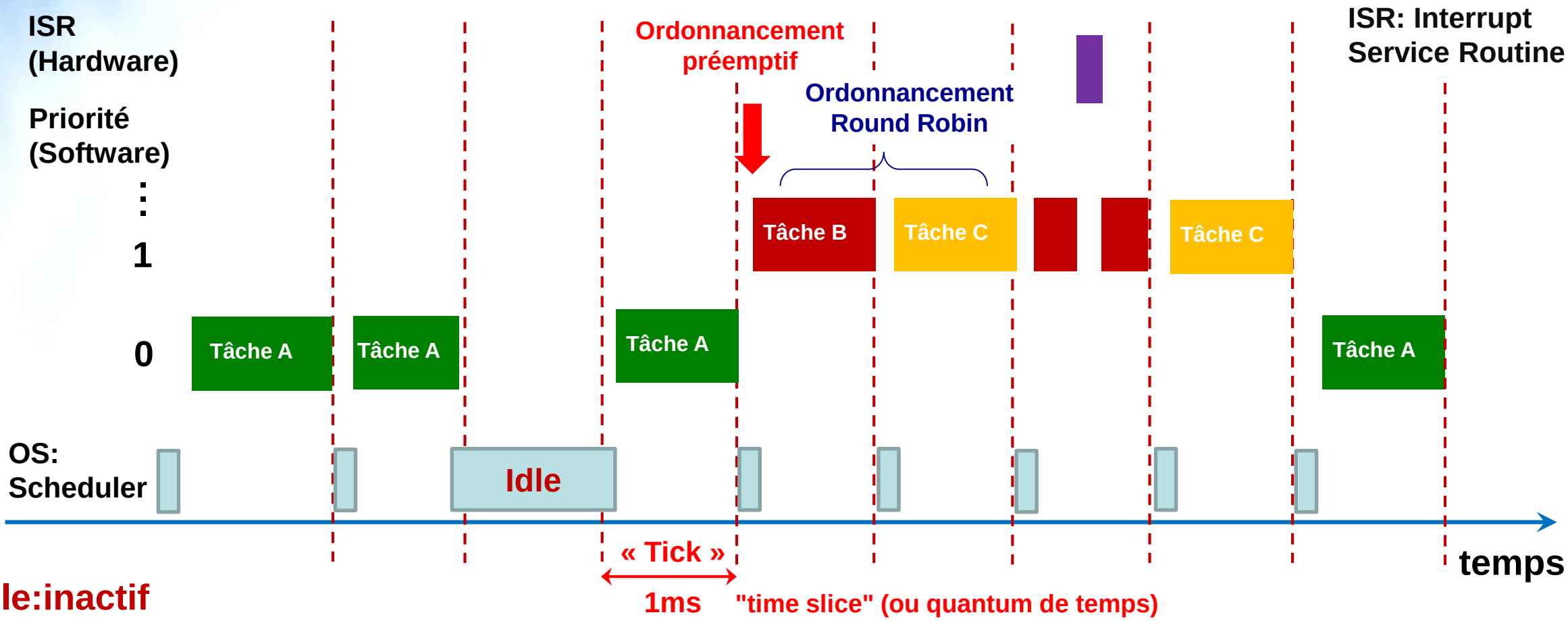
4.3. L'ordonnanceur (Scheduler) :

- L'ordonnanceur de FreeRTOS est responsable de la gestion des tâches et de l'allocation du CPU.
- Il peut être configuré pour utiliser différents algorithmes d'ordonnancement, comme le préemptif (où une tâche de haute priorité peut interrompre une tâche de basse priorité) ou le coopératif.



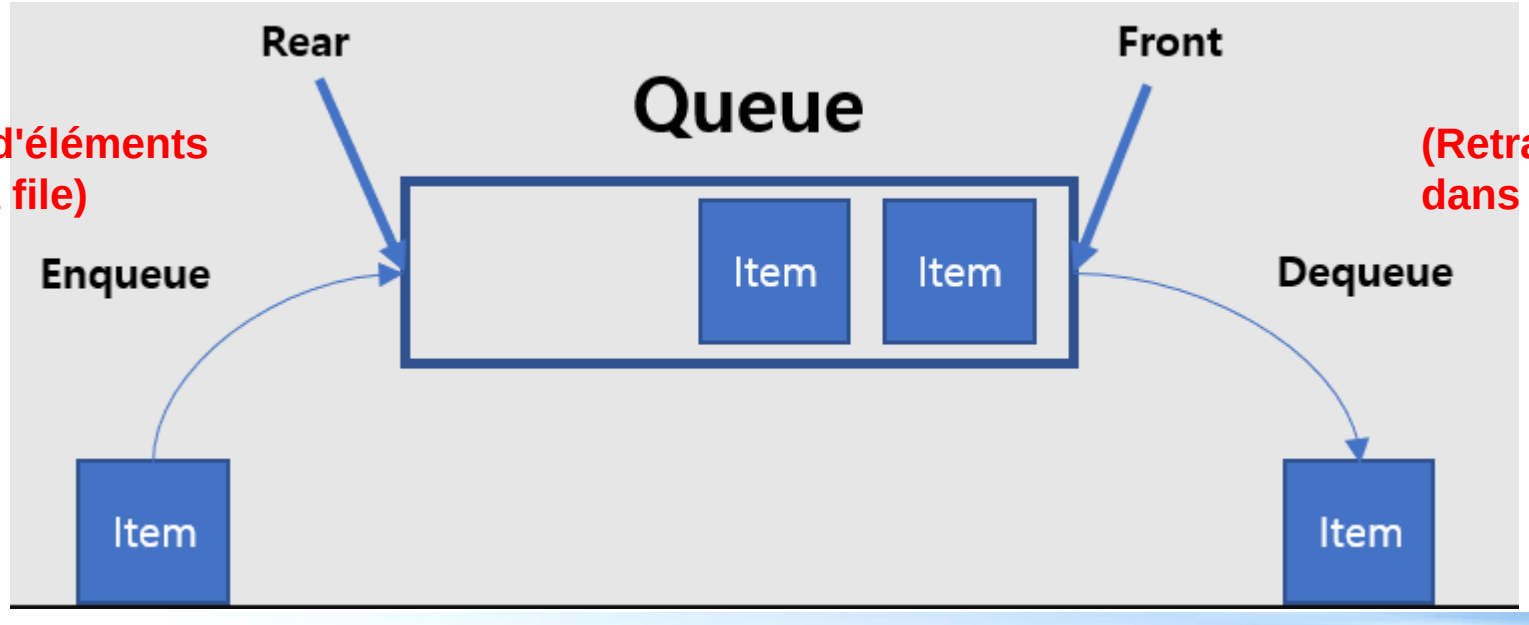
- L'ordonnanceur est démarré avec la fonction
  - **vTaskStartScheduler()** avec **sous Arduino IDE**
  - **osKernelStart() sous STM32CubeIDE**
- FreeRTOS utilise principalement un algorithme de planification de priorité fixe préemptive avec une planification Round-Robin (RR) entre les tâches de même priorité.

Exemple d'ordonnancement FreeRTOS (cas d'un seul noyau):



4.4. Les Files d'attente (Queues) :

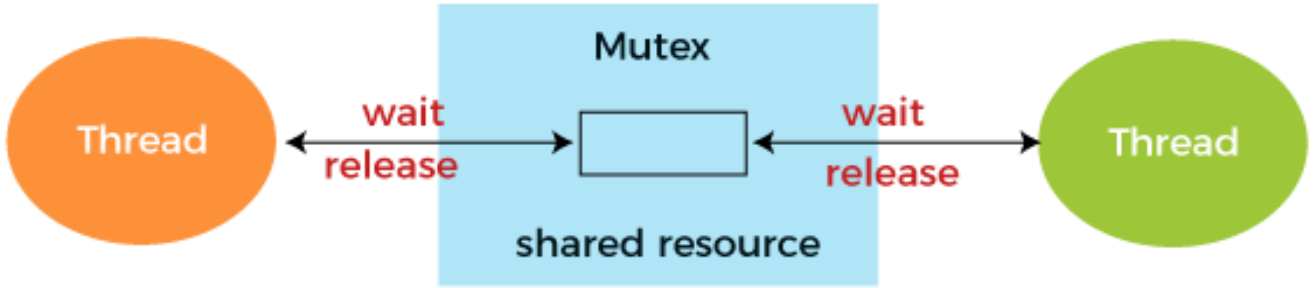
- Les files d'attente permettent la communication entre les tâches et entre les tâches et les interruptions.
- Elles sont utilisées pour transmettre des données de manière sécurisée entre les tâches.



- Les files d'attente sont créées avec la fonction:
  - **xQueueCreate() sous Arduino IDE**
  - **osMessageQueueNew sous STM32CubeIDE**
- Elles sont manipulées avec des fonctions comme
  - **xQueueSend()** et **xQueueReceive() sous Arduino IDE**
  - **osMessageQueuePut()** et **osMessageQueueGet() sous STM32CubeIDE**

4.5. Les Mutexes (MUTual EXclusion) :

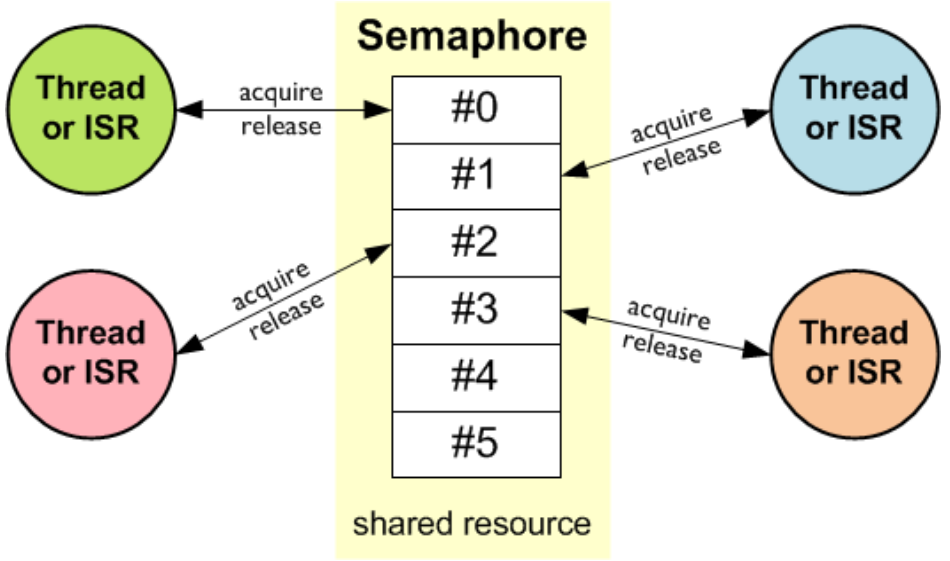
- Les mutex sont utilisés pour protéger l'accès à une ressource partagée.
- Un mutex est utilisé pour garantir qu'une seule tâche ou thread accède à une ressource partagée à la fois.



- Un jeton mutex est créé puis passé entre les threads (ils peuvent acquérir et libérer le mutex).
- Un jeton mutex est binaire et limité, c'est-à-dire qu'il est soit disponible, soit bloqué par un thread propriétaire

4.6. Les Sémaphores :

- Les sémaphores sont utilisés pour gérer et protéger l'accès aux ressources partagées.
- Alors qu'un Mutex permet à un seul thread d'accéder à une ressource partagée à la fois, un sémaphore peut être utilisé pour permettre à un nombre fixe de threads d'accéder à des ressources partagées.



- Il existe deux catégories de Sémaphore:
  - *Sémaphore de comptage (**Counting semaphore**): Peut contenir un nombre entier positif quelconque de jetons. Lorsque le compteur du sémaphore passe à 0, cela signifie que les processus occupent toutes les ressources.*
  - *sémaphore binaire (**Binary semaphore**): Ne peut contenir que deux états (0 ou 1)*

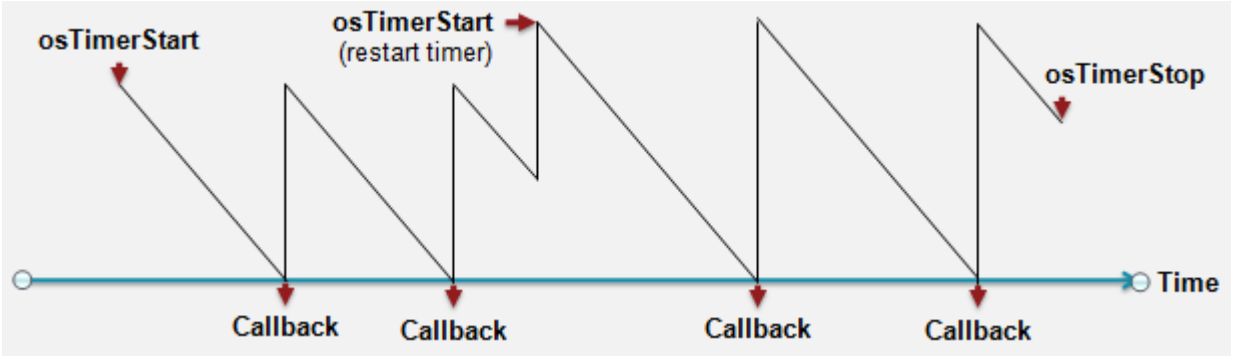
- Les sémaphores sont créés avec
  - **xSemaphoreCreateBinary()**, **xSemaphoreCreateCounting()**, etc. **sous Arduino IDE**
  - **osSemaphoreNew() sous STM32CubeIDE**
- Elle sont manipulées avec
  - **xSemaphoreTake()** et **xSemaphoreGive() sous Arduino IDE**
  - **osSemaphoreAcquire()** et **osSemaphoreRelease() sous STM32CubeIDE**

Exemple: Imaginons un scénario où plusieurs tâches doivent utiliser **un ensemble de 3 imprimantes partagées**. Chaque imprimante peut être utilisée par une tâche à la fois, et les autres tâches doivent attendre si toutes les imprimantes sont occupées.

1. **Création de la sémaphore** : **xSemaphoreCreateCounting(3, 3)** :
   - **3** : Maximum de ressources disponibles (3 imprimantes).
   - **3** : Compteur initial (au début, toutes les imprimantes sont libres).
2. **Prise de la sémaphore: xSemaphoreTake** :
   - Lorsqu'une tâche veut utiliser une imprimante, elle décrémente le compteur.
   - Si le compteur est à **0** (toutes les imprimantes sont occupées), la tâche attend jusqu'à ce qu'une imprimante soit libérée.
3. **Libération de la sémaphore: xSemaphoreGive** :
   - Quand une tâche termine son travail avec l'imprimante, elle incrémente le compteur pour indiquer qu'une imprimante est libre.

4.7. Les Timers :

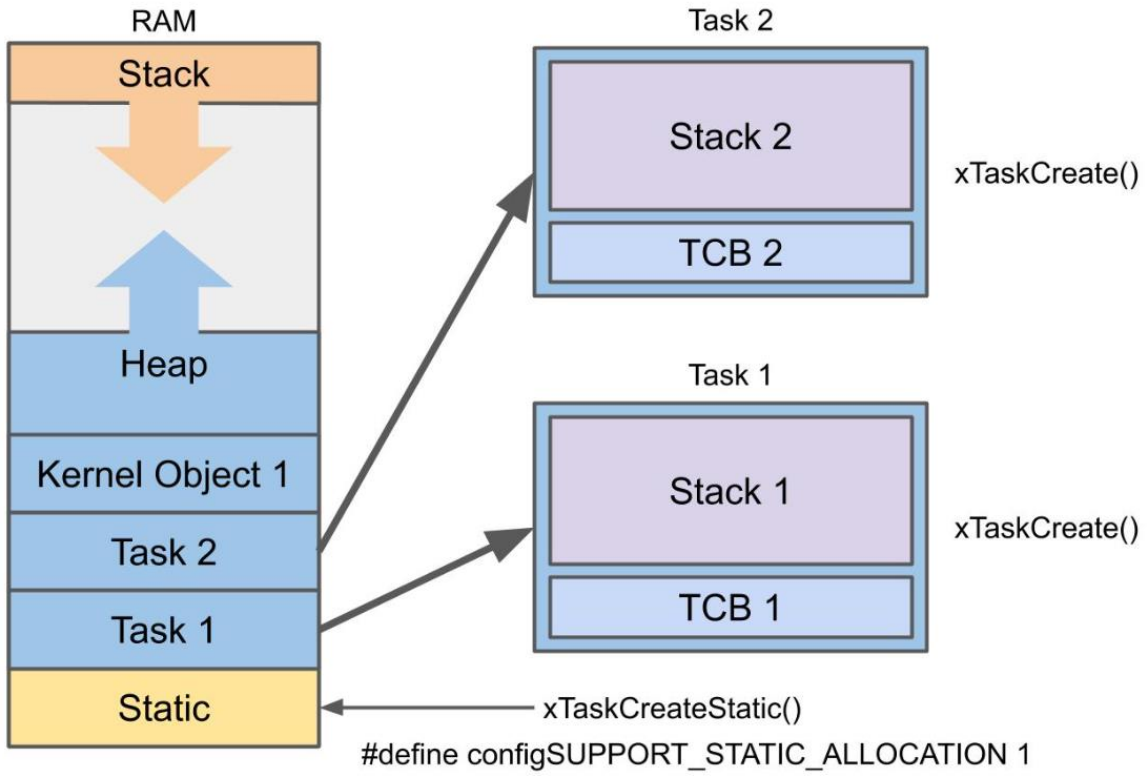
- FreeRTOS fournit des minuteries logicielles (Timers) qui peuvent être utilisées pour exécuter du code à des intervalles réguliers ou après un certain délai.
- Les Timers sont créés avec
  - **xTimerCreate() sous Arduino IDE**
  - **osTimerNew() sous STM32CubeIDE**



- Il sont contrôlés avec des fonctions comme **xTimerStart()** et **xTimerStop()**.

4.8. Gestion de la mémoire :

- FreeRTOS utilise différents schémas de gestion de la mémoire pour allouer et libérer la mémoire pour les tâches et les autres objets dynamiques.
- Il propose plusieurs allocateurs de mémoire, comme **heap_1**, **heap_2**, **heap_3**, **heap_4** et **heap_5**, chacun ayant des caractéristiques spécifiques.



4.9. Exemple de Création et Gestion d'une Tâche avec Arduino IDE:

```
#include "FreeRTOS.h"
#include "task.h"

// Fonction de la tâche
void vTaskFunction(void *pvParameters) {
    for (;;) {
        // Code de la tâche
        vTaskDelay(pdMS_TO_TICKS(1000)); // Délai de 1000 ms
    }
}

int main(void) {
    // Création de la tâche
    xTaskCreate(
        vTaskFunction,          // Pointeur vers la fonction de la tâche
        "Task 1",               // Nom de la tâche (pour le débogage)
        configMINIMAL_STACK_SIZE, // Taille de la pile de la tâche
        NULL,                   // Paramètre passé à la tâche
        tskIDLE_PRIORITY,       // Priorité de la tâche
        NULL                    // Pointeur vers le handle de la tâche (optionnel)
    );

    // Démarrage de l'ordonnanceur
    vTaskStartScheduler();

    // Le programme ne doit jamais atteindre ici
    for (;;);
}
```

5. Noyau FreeRTOS et Documentation

Le site de FreeRTOS est une ressource essentielle pour toute personne souhaitant utiliser FreeRTOS,

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

Il offre des guides, de la documentation, des téléchargements et du support pour les développeurs de systèmes embarqués

Télécharger FreeRTOS: https://www.freertos.org/a00104.html

FreeRTOS 202212.01

Package containing the FreeRTOS Kernel, FreeRTOS-Plus libraries and AWS IoT libraries, along with example projects. Source code is also available on GitHub. Separately, the latest FreeRTOS Kernel can also be downloaded from here.

Download

Documentation:

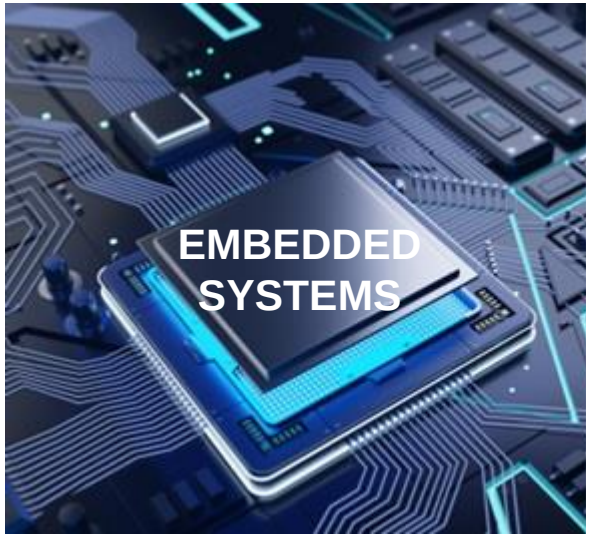
https://www.freertos.org/Documentation/RTOS_book.html

- **Mastering the FreeRTOS Real Time Kernel - A Hands On Tutorial Guide:**

  Latest version of the book on GitHub

PDF Version	Companion Example Source	Kernel Version Referenced
Release Version - 1.0 Edition	Feb-07-2024	FreeRTOS V10.6.2
Pre-release Edition	V9.0.0	FreeRTOS V8.x.x

- **FreeRTOS Reference Manual:**
  - version 10.0.0 issue 1



FIN !
Questions ?