

Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique



University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

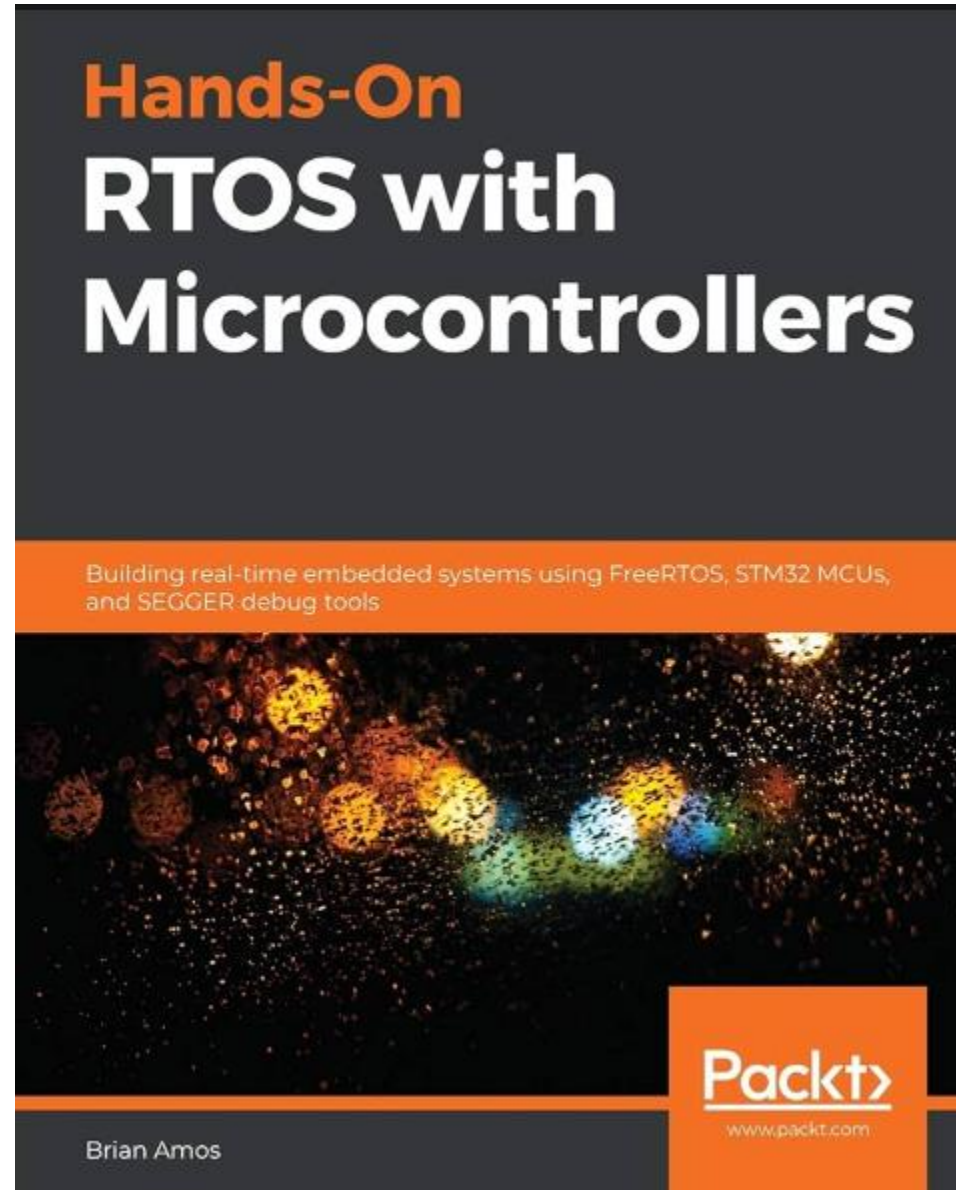
CEG4566/CSI4541/SEG4545

Conception de systèmes informatiques en temps réel

Hiver 2024

Professeur : Mohamed Ali Ibrahim, ing., Ph.D.

Source :



Chapitre 2 :

Comprendre les tâches du RTOS

Plan

- Introduction à la programmation en super-boucles
- Réaliser des opérations parallèles avec des super-boucles
- Comparaison entre les tâches RTOS et les super-boucles
- Réaliser des opérations parallèles avec des tâches RTOS
- Tâches RTOS et super boucles
 - avantages et inconvénients

Introduction à la programmation en super-boucles

- Il existe une propriété commune à tous les systèmes embarqués
 - ils n'ont pas de point de sortie.
- En raison de sa nature, le code intégré est généralement censé être toujours disponible
 - s'exécutant silencieusement en arrière-plan, s'occupant des tâches ménagères, et prêt à recevoir à tout moment les données de l'utilisateur.
- Contrairement aux environnements de bureau qui sont conçus pour démarrer et arrêter des programmes, un microcontrôleur n'a rien à faire s'il quitte la fonction `main()`.

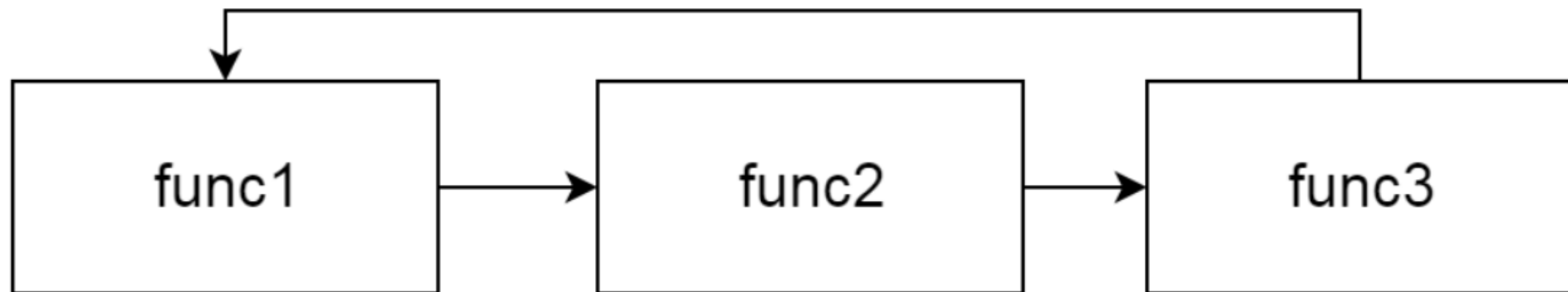
Le code suivant représente l'idée de base d'une super boucle

```
void main ( void )  
{  
    while(1)  
    {  
        func1() ;  
        func2() ;  
        func3() ;  
        //font des choses utiles, mais ne renvoient rien  
        //(sinon, où irions-nous... que ferions-nous... ?!)  
    }  
}
```

Basic Super Loop Setup



Brian Amos

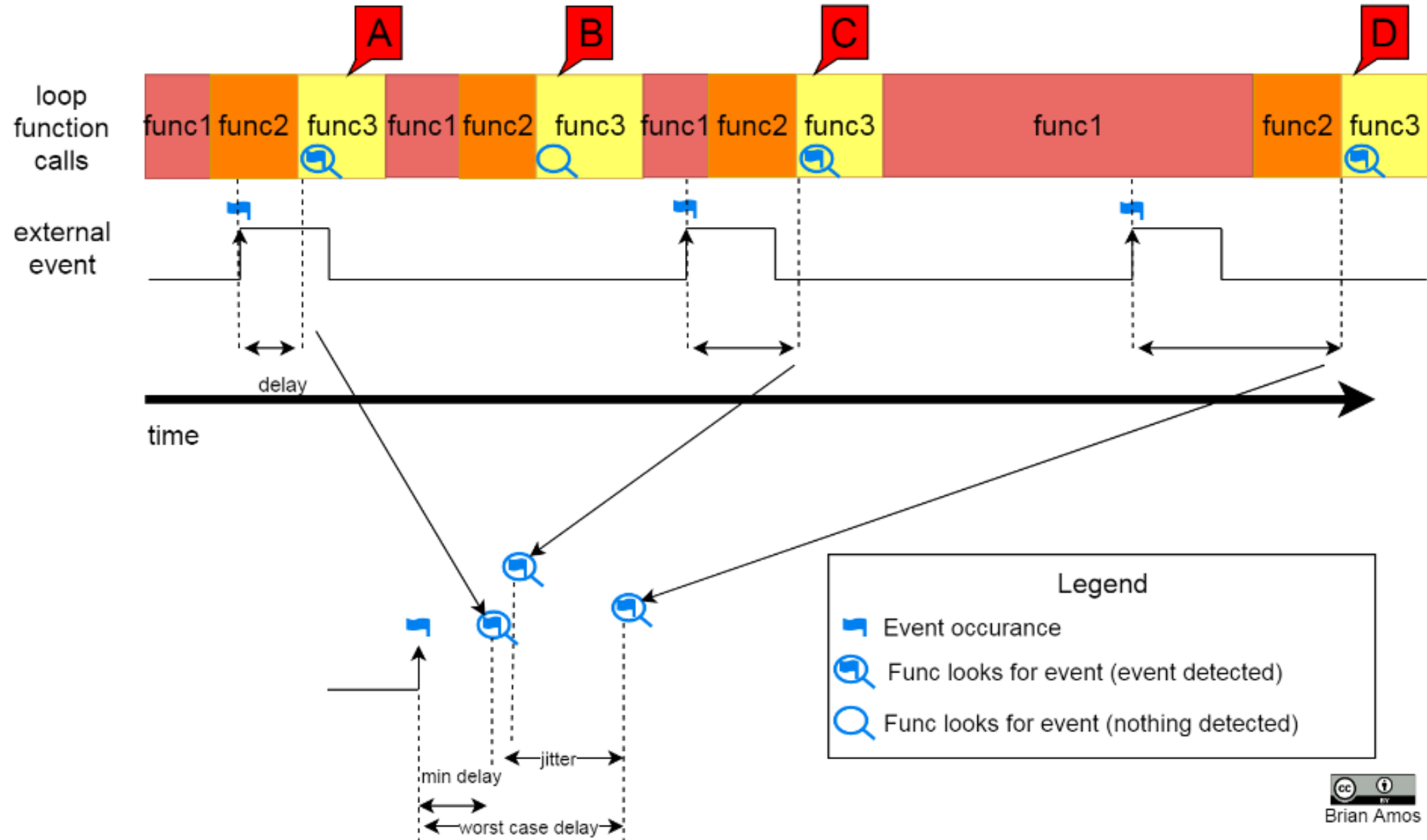


Super Loop

Super boucles dans les systèmes en temps réel (1/2)

- Lorsque des super boucles simples fonctionnent rapidement (généralement parce qu'elles ont une fonctionnalité/responsabilité limitée), elles sont assez réactives.
- Cependant, la simplicité de la super boucle peut être bonne et mauvaise.
- Comme chaque fonction suit toujours la fonction précédente, elles sont toujours appelées dans le même ordre et dépendent entièrement les unes des autres.
- Tout retard introduit par une fonction se propage à la fonction suivante, ce qui augmente le temps total nécessaire à l'exécution de cette itération de la boucle (comme le montre le diagramme suivant).
- Si **func1** met 10 ms à s'exécuter une fois dans la boucle, puis 100 ms la fois suivante, **func2 ne sera** pas appelé aussi rapidement la deuxième fois qu'il l'a été la première fois.

Basic Super Loop Execution and Jitter



Super boucles dans les systèmes en temps réel (2/2)

- Dans le schéma précédent, **func3** est chargé de vérifier l'état d'un drapeau représentant un événement externe (cet événement signale un front montant d'un signal).
- La fréquence à laquelle **func3** vérifie l'indicateur dépend du temps d'exécution de **func1** et **func2**.
- Une super-boucle bien conçue et réactive s'exécute généralement très rapidement, en vérifiant les événements plus souvent qu'ils ne se produisent (point B).
- Lorsqu'un événement externe se produit, la boucle ne le détecte pas avant la prochaine exécution de **func3** (points A, C et D).
- Notez qu'il y a un délai entre le moment où l'événement est généré et celui où il est détecté par **func3**.
- Notez également que le délai n'est pas toujours cohérent :
 - cette différence de temps est appelée gigue.

Super boucles dans les systèmes en temps réel (2/2)

- Si un système a une quantité maximale connue de gigue lorsqu'il répond à un événement, il est considéré comme déterministe.
- C'est-à-dire qu'il répondra de manière fiable à un événement dans le délai spécifié après que cet événement se soit produit.
- Un niveau élevé de déterminisme est crucial pour les composants critiques d'un système en temps réel car, sans lui, le système pourrait ne pas réagir à temps à des événements importants.

Réaliser des opérations parallèles avec des super-boucles

- Même si une super boucle de base ne peut parcourir les fonctions que de manière séquentielle, il existe encore des moyens de réaliser le parallélisme.
- Les MCU disposent de différents types de matériel spécialisé conçus pour soulager l'unité centrale, tout en permettant un système très réactif.
- Cette section présente ces systèmes et la manière dont ils peuvent être utilisés dans le contexte d'un programme de type super-boucle.

Introduction des interruptions (1/2)

- L'interrogation pour un seul événement n'est pas seulement un gaspillage en termes de cycles de l'unité centrale et d'énergie
 - il en résulte également un système qui ne réagit à rien d'autre, ce qui devrait généralement être évité.
- Alors, comment faire en sorte qu'un processeur à cœur unique puisse effectuer des tâches en parallèle ?
- Nous ne pouvons pas
 - il n'y a qu'un seul processeur après tout. Mais comme notre processeur est susceptible d'exécuter des millions d'instructions par seconde, il est possible de lui faire exécuter des tâches qui sont assez proches du parallélisme.
- Les MCU comprennent également un matériel dédié à la génération d'interruptions.
- Les interruptions fournissent des signaux au MCU qui lui permettent de passer directement à une routine de service d'interruption (ISR) lorsque l'événement se produit.

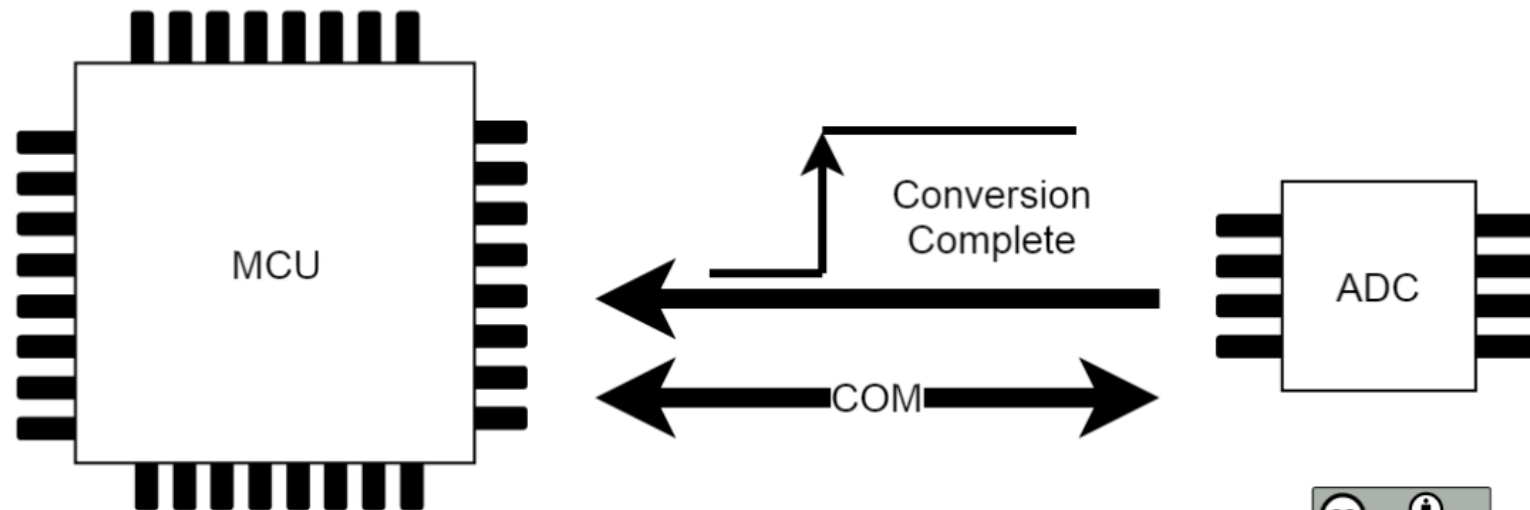
Introduction des interruptions (2/2)

- Cette fonctionnalité est tellement essentielle que les cœurs ARM Cortex-M fournissent un périphérique standardisé, appelé nested vector interrupt controller (NVIC).
- Le NVIC fournit un moyen commun de traiter les interruptions.
- La partie imbriquée de ce terme signifie que même les interruptions peuvent être interrompues par d'autres interruptions ayant une priorité plus élevée.
- C'est très pratique, car cela nous permet de minimiser la latence et la gigue pour les éléments du système les plus critiques en termes de temps.
- Alors, comment les interruptions s'intègrent-elles dans une super-boucle de manière à mieux créer l'illusion d'une activité parallèle ?
- Le code à l'intérieur d'un ISR est généralement aussi court que possible, afin de minimiser le temps passé dans l'interruption.

Exemple de convertisseur analogique-numérique externe (ADC)

- Dans le matériel ADC, une broche est dédiée à la signalisation que la lecture d'une valeur analogique a été convertie en une représentation numérique et qu'elle est prête à être transférée au MCU.
- Le MCU lance alors un transfert sur le support de communication (COM dans le diagramme).

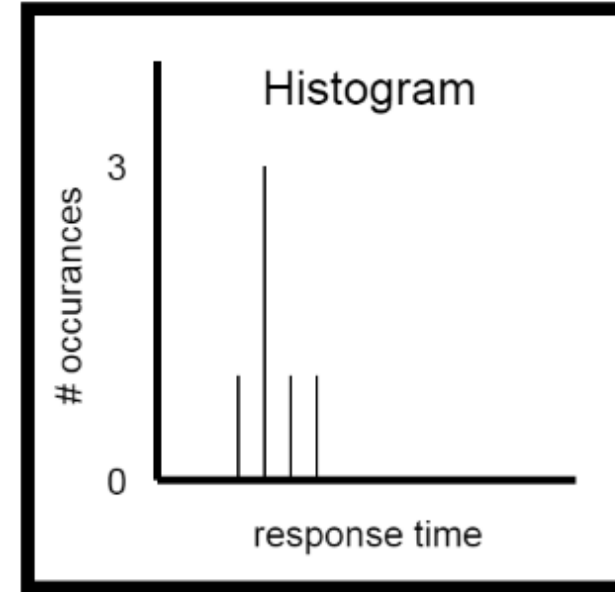
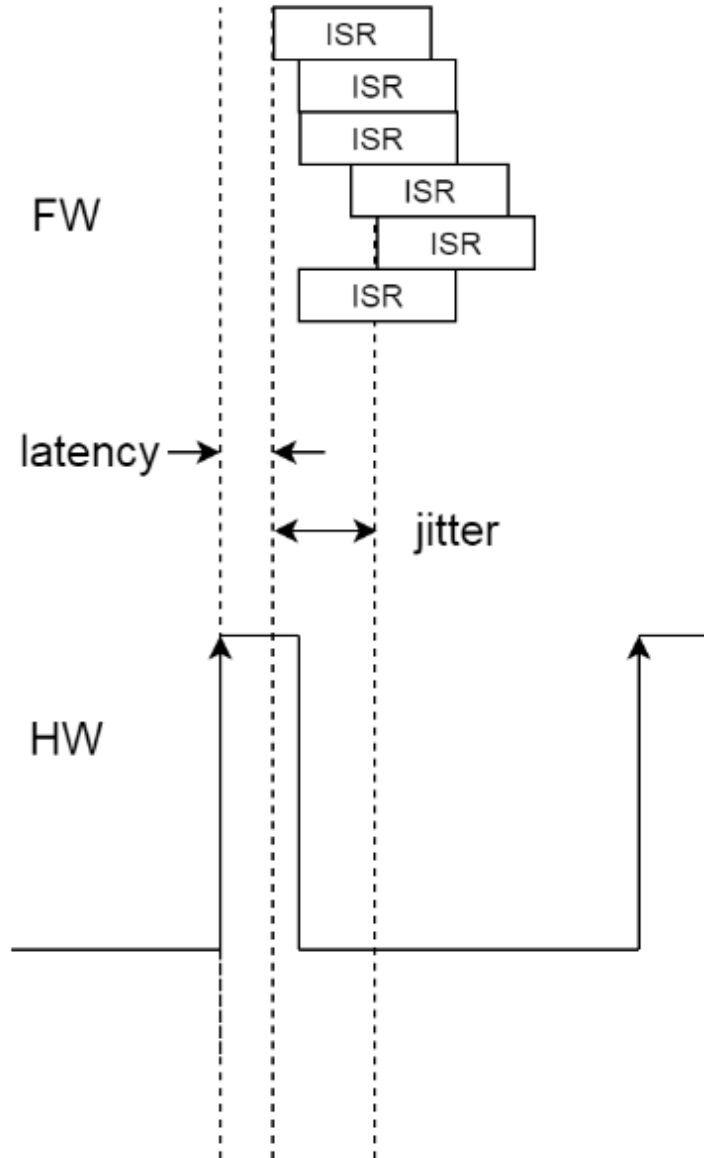
MCU with External ADC



ISR Gigue répondant au front montant (1/2)

- Le diagramme suivant montre (diapositive suivante) six exemples différents d'ISR appelés en réponse à un front montant d'un signal.
- Le peu de temps qui s'écoule entre le moment où le front montant se produit dans le matériel et le moment où l'ISR est invoqué dans le microprogramme est la latence minimale.
- La gigue dans la réponse de l'ISR est la différence de latence sur plusieurs cycles différents.

ISR Jitter Responding to Rising Edge



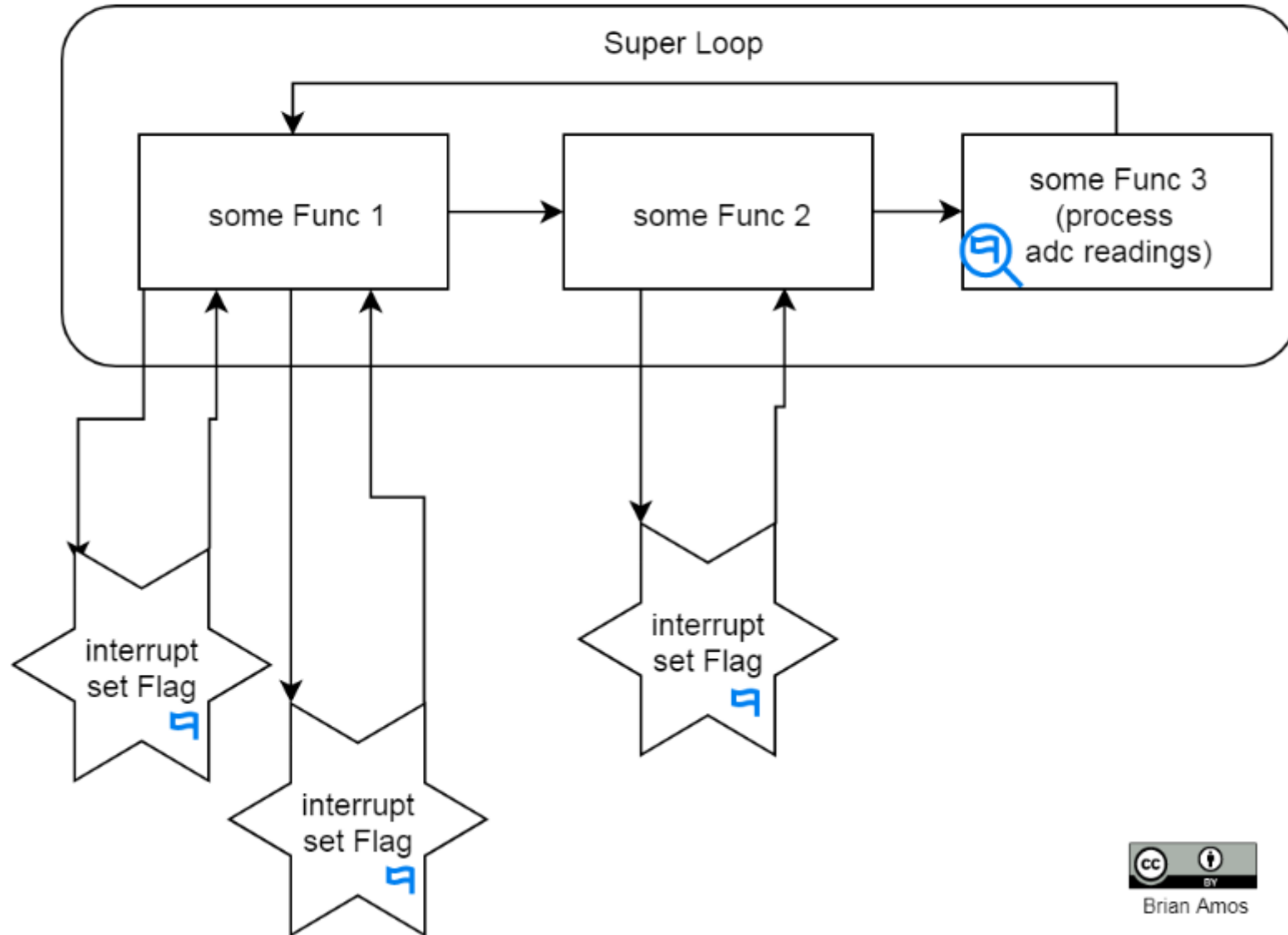
Latence et gigue pour les ISR critiques

- Il existe différentes manières de minimiser la latence et la gigue pour les ISR critiques.
- Dans les MCU ARM Cortex-M, les priorités d'interruption sont flexibles
 - une même source d'interruption peut se voir attribuer différentes priorités au moment de l'exécution.
- La possibilité de redéfinir les priorités des interruptions est un moyen de s'assurer que les parties les plus importantes d'un système bénéficient de l'unité centrale lorsqu'elles en ont besoin.
- Lorsque plusieurs interruptions sont imbriquées, elles ne renvoient pas entièrement les informations suivantes
 - Les processeurs ARM Cortex M disposent d'une fonction utile appelée "interrupt-tail chaining" (enchaînement d'interruptions et de queues).
- Si le processeur détecte qu'une interruption est sur le point de se terminer, mais qu'une autre est en attente, l'ISR suivant sera exécuté sans que le processeur ne rétablisse totalement l'état antérieur à l'interruption, ce qui réduit encore le temps de latence.

Interruptions et super boucles

- Une façon de minimiser les instructions et la responsabilité dans l'ISR est d'effectuer la plus petite quantité de travail possible à l'intérieur de l'ISR, puis d'activer un drapeau qui sera vérifié par le code s'exécutant dans la super-boucle.
- De cette manière, l'interruption peut être traitée dès que possible, sans que l'ensemble du système ne soit consacré à l'attente de l'événement.
- Dans le diagramme suivant (diapositive suivante), vous remarquerez que l'interruption est générée plusieurs fois avant d'être traitée par **func3**.
- En fonction de l'objectif exact de cette interruption, elle prend généralement une valeur du périphérique associé et la place dans un tableau (ou prend une valeur d'un tableau et l'envoie dans les registres du périphérique).

Basic Super Loop with Interrupts

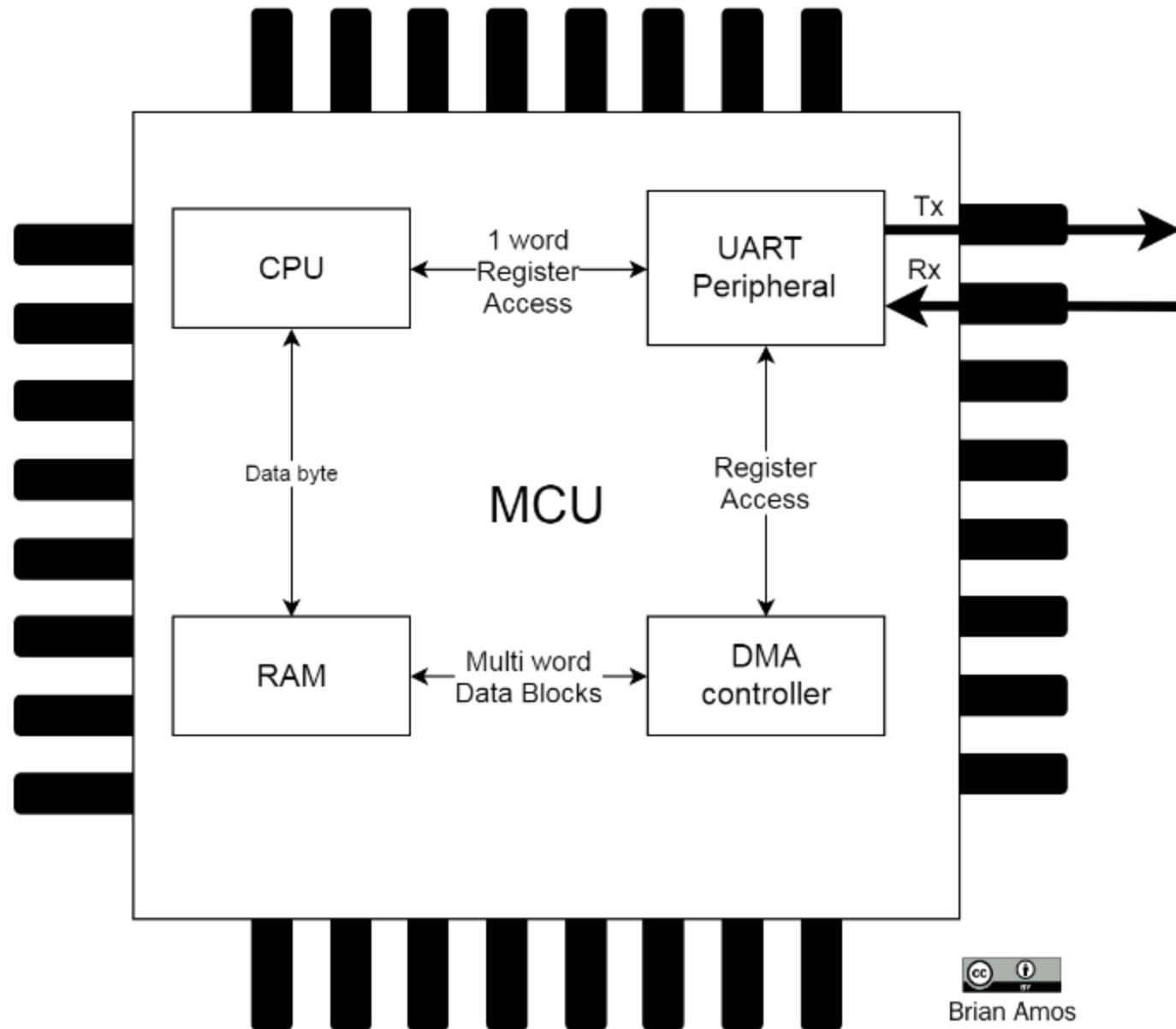


Brian Amos

Présentation du DMA

- Vous souvenez-vous de l'affirmation selon laquelle le processeur ne pouvait pas vraiment faire des choses en parallèle ?
 - C'est toujours le cas.
- Cependant, les MCU modernes contiennent plus qu'un simple noyau de traitement.
- Pendant que notre noyau de traitement traite les instructions, de nombreux autres sous-systèmes matériels sont à l'œuvre à l'intérieur de l'unité centrale de commande.
- L'un de ces sous-systèmes très performants est appelé **contrôleur d'accès direct à la mémoire (DMA)**.

DMA Paths in MCUs (1/3)



Chemins DMA dans les MCU (2/3)

- Le diagramme précédent présente un schéma fonctionnel matériel très simplifié qui montre une vue de deux chemins de données différents disponibles à partir de la RAM vers un périphérique UART.
- Dans le cas de la réception d'un flux d'octets en provenance d'un UART sans DMA, les informations de l'UART sont transférées dans les registres de l'UART, lues par l'unité centrale, puis transférées dans la mémoire vive pour y être stockées :
 1. L'unité centrale doit détecter la réception d'un octet (ou d'un mot), soit en interrogeant les drapeaux du registre UART, soit en mettant en place une routine d'interruption de service qui sera déclenchée lorsqu'un octet sera prêt.
 2. Une fois l'octet transféré depuis l'UART, l'unité centrale peut le placer dans la mémoire vive en vue d'un traitement ultérieur.
 3. Les étapes 1 et 2 sont répétées jusqu'à ce que l'ensemble du message soit reçu.

Chemins DMA dans les MCU (3/3)

- Lorsque le DMA est utilisé dans le même scénario, il se produit ce qui suit :
 1. L'unité centrale configure le contrôleur DMA et les périphériques pour le transfert.
 2. Le contrôleur DMA se charge de TOUS les transferts entre le périphérique UART et la RAM. Cela ne nécessite aucune intervention de la part du CPU.
 3. L'unité centrale sera informée de la fin du transfert et pourra traiter directement l'ensemble du flux d'octets.

Mise à l'échelle d'une super boucle (1/4)

- Nous avons donc maintenant un système réactif qui peut traiter les interruptions de manière fiable.
- Nous avons peut-être configuré un contrôleur DMA pour qu'il s'occupe également des tâches lourdes pour les périphériques de communication.
- Pourquoi avons-nous besoin d'un RTOS ?
- Il est tout à fait possible que vous ne le sachiez pas !
- Si le système traite un nombre limité de responsabilités et qu'aucune d'entre elles n'est particulièrement compliquée ou ne prend beaucoup de temps, il n'est peut-être pas nécessaire d'avoir recours à quelque chose de plus sophistiqué qu'une super-boucle.

Mise à l'échelle d'une super boucle (2/4)

- Toutefois, si le système est également responsable de la création d'une interface utilisateur (IU), de l'exécution d'algorithmes complexes qui prennent du temps, ou de la gestion de piles de communication complexes, il est très probable que ces tâches prendront un temps non négligeable.
- Si une interface utilisateur accrocheuse avec beaucoup d'animations commence à piétiner un peu parce que l'unité MCU doit collecter des données à partir d'un capteur critique, ce n'est pas grave.
- L'animation peut être réduite ou éliminée, et la partie importante du système en temps réel reste intacte.
- Mais, que se passe-t-il si l'animation est toujours parfaite, même si le capteur n'a pas transmis certaines données ?

Mise à l'échelle d'une super boucle (3/4)

- Ce problème se pose chaque jour de différentes manières dans notre secteur.
- Parfois, si le système a été suffisamment bien conçu, les données manquantes sont détectées et signalées (mais elles ne peuvent pas être récupérées : elles sont perdues à jamais).
- Si l'équipe de conception a de la chance, il se peut même qu'elle ait échoué de cette manière lors des tests internes.
- Cependant, dans de nombreux cas, les données manquantes des capteurs passent totalement inaperçues jusqu'à ce que quelqu'un remarque que l'un des relevés semble un peu erroné... parfois.
- Si tout le monde a de la chance, le rapport de bogue pour la lecture approximative pourrait inclure une indication que cela ne semble se produire que lorsque quelqu'un se trouve sur le panneau avant (en train de jouer avec ces animations fantaisistes).
- Cela donnerait au moins un indice au pauvre ingénieur en microprogrammation chargé de déboguer le problème, mais nous n'avons souvent pas cette chance.

Mise à l'échelle d'une super boucle (4/4)

- Ce sont les types de systèmes pour lesquels un RTOS est nécessaire.
- L'un des points forts des ordonnanceurs préemptifs est de garantir que les tâches les plus urgentes sont toujours exécutées lorsque c'est nécessaire et de programmer l'exécution des tâches moins prioritaires dès qu'il reste du temps disponible.
- Dans ce type de configuration, les relevés critiques des capteurs pourraient faire l'objet d'une tâche distincte et se voir attribuer une priorité élevée
 - interrompant effectivement tout le reste du système (à l'exception des ISR) lorsqu'il était temps de s'occuper du capteur.
- Cette pile de communication complexe pourrait se voir attribuer une priorité inférieure à celle du capteur critique.
- Enfin, l'interface utilisateur clinquante avec ses animations fantaisistes s'approprie les cycles processeurs restants.
- Il est libre d'effectuer autant d'animations d'alpha-blending qu'il le souhaite, mais uniquement lorsque le processeur n'a rien d'autre à faire.

Comparaison entre les tâches RTOS et les super-boucles

- Jusqu'à présent, nous n'avons mentionné les tâches que de manière très superficielle, mais qu'est-ce qu'une tâche ?
- Il est facile de considérer une tâche comme une autre boucle principale.
- Dans un RTOS préemptif, il existe deux différences principales entre les tâches et les super-boucles :
 - Chaque tâche reçoit sa propre pile privée.
 - Contrairement à une super boucle dans main, qui partageait la pile du système, les tâches reçoivent leur propre pile qu'aucune autre tâche du système n'utilisera.
 - Cela permet à chaque tâche d'avoir sa propre pile d'appels sans interférer avec les autres tâches.
 - Une priorité est attribuée à chaque tâche.
 - Cette priorité permet à l'ordonnanceur de décider quelle tâche doit être exécutée (l'objectif est de s'assurer que la tâche la plus prioritaire du système effectue toujours un travail utile).

Réaliser des opérations parallèles avec des tâches RTOS

- Plus tôt, nous avons examiné une super boucle qui passait par trois fonctions.
- Maintenant, pour un exemple très simple, déplaçons chacune des trois fonctions dans sa propre tâche.
- Ces trois tâches simples nous permettront d'examiner les points suivants :
 - **Modèle théorique de programmation des tâches** : Comment les trois tâches peuvent être décrites théoriquement
 - **Ordonnancement round-robin réel** : aspect des tâches lorsqu'elles sont exécutées à l'aide d'un algorithme d'ordonnancement round-robin
 - **Ordonnancement préemptif réel** : Les tâches sont exécutées à l'aide d'un ordonnancement préemptif.
- Dans les programmes du monde réel, il n'y a presque jamais une seule fonction par tâche ; nous n'utilisons cela que comme une analogie à la super boucle trop simpliste de tout à l'heure.

Modèle théorique de programmation des tâches

- Voici un pseudo-code qui utilise une super boucle pour exécuter trois fonctions.
- Les trois mêmes fonctions sont également incluses dans un système basé sur les tâches
 - chaque tâche RTOS (à droite) contient les mêmes fonctionnalités que les fonctions de la super boucle à gauche.
- Nous nous en servons à l'avenir pour discuter des différences dans la manière dont le code est exécuté lors de l'utilisation d'une super boucle par rapport à l'utilisation d'une approche axée sur les tâches avec un ordonnanceur (diapositive suivante).

Super Loop

```
func1()
{
    //functionality 1
}

func2()
{
    //functionality 2
}

func3()
{
    //functionality 3
}

main()
{
    while(1)
    {
        func1();
        func2();
        func3();
    }
}
```

RTOS Tasks

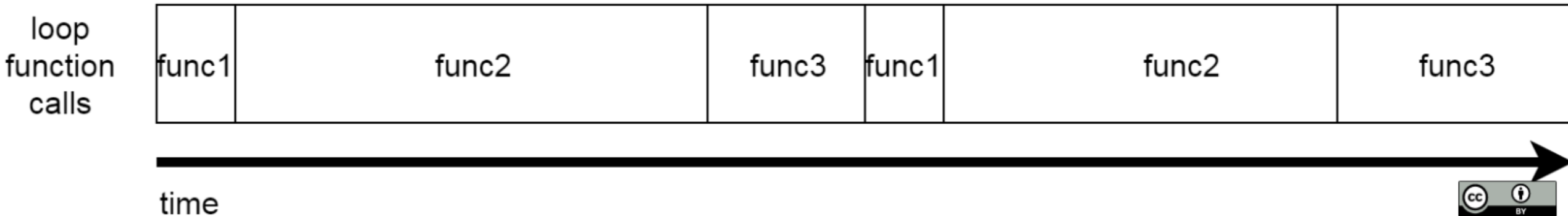
```
Task1()
{
    while(1)
    {
        //functionality of task 1
    }
}

Task2()
{
    while(1)
    {
        //functionality of task 2
    }
}

Task3()
{
    while(1)
    {
        //functionality of task 3
    }
}

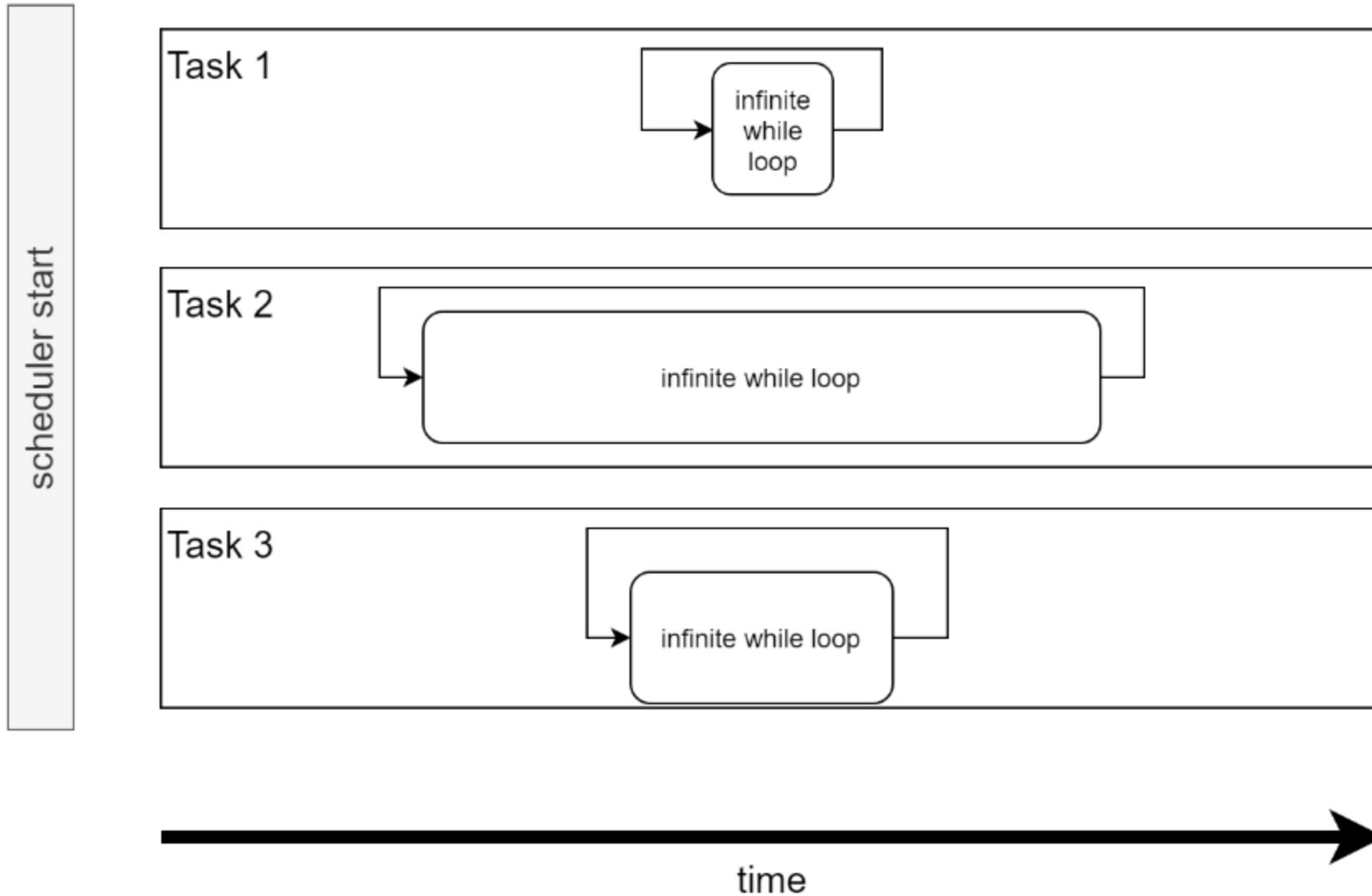
main()
{
    createTask1(&Task1);
    createTask2(&Task2);
    createTask3(&Task3);
    StartScheduler(); //never returns
}
```

Super Loop Execution Unrolled over Time



Brian Amos

Basic Task Setup (Programming Model)



Brian Amos

Configuration des tâches de base (modèle de programmation)

- Dans le diagramme, vous remarquerez que la taille de chaque boucle while n'est pas la même.
- C'est l'un des nombreux avantages de l'utilisation d'un ordonnanceur qui exécute les tâches en parallèle par rapport à une super boucle.
 - le programmeur n'a pas besoin de se préoccuper immédiatement du fait que la longueur de la boucle la plus longue ralentit les autres boucles plus étroites.
- Le diagramme montre que la **tâche 2** a une boucle beaucoup plus longue que la **tâche 1**.
- Dans un système de super boucle, la fonctionnalité de func1 s'exécuterait moins souvent (puisque la super boucle devrait exécuter **func1**, puis **func2**, puis **func3**).
- Dans un modèle de programmation basé sur les tâches, ce n'est pas le cas
 - la boucle de chaque tâche peut être considérée comme isolée des autres tâches du système
 - et ils fonctionnent tous en parallèle.

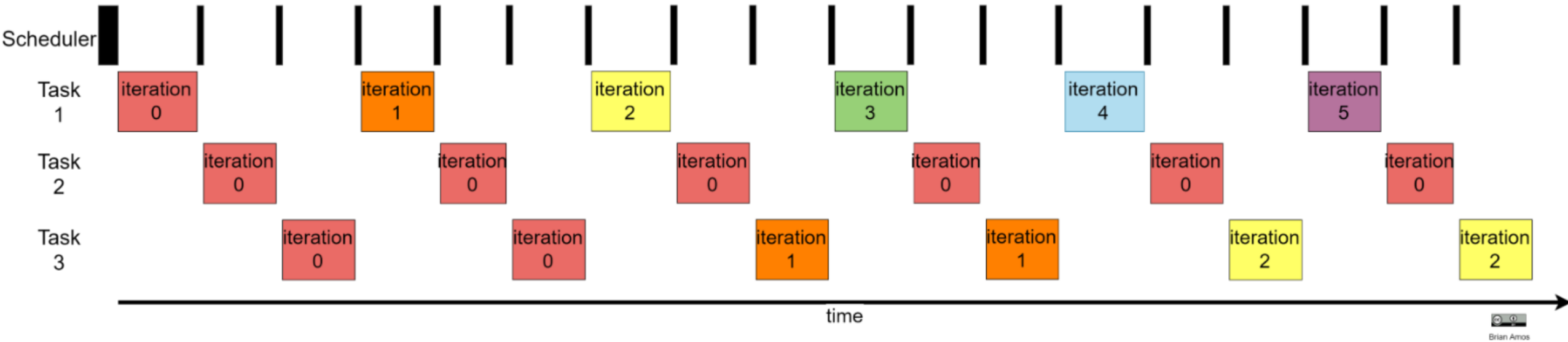
Ordonnancement à la ronde (1/5)

- L'une des façons les plus simples de conceptualiser l'exécution réelle d'une tâche est d'utiliser l'ordonnancement round-robin.
- Dans l'ordonnancement round-robin, chaque tâche dispose d'une petite tranche de temps pour utiliser le processeur, ce qui est contrôlé par l'ordonnanceur.
- Tant que la tâche a du travail à effectuer, elle s'exécute.
- En ce qui concerne la tâche, le processeur est entièrement à sa disposition.
- Le ordonnanceur prend en charge toute la complexité du passage au contexte approprié pour la tâche suivante (voir diapositive suivante).

Ordonnancement à la ronde (2/5)

For section (2/5) of the course, this isolation and perceived parallel execution are some of the benefits of using an RTOS; it alleviates some of the complexity for the programmer.

Basic Task Setup
(Round Robin Scheduling)



Ordonnancement à la ronde (3/5)

- Il s'agit des trois mêmes tâches que celles présentées précédemment, sauf qu'au lieu d'une conceptualisation théorique, chaque itération dans les boucles des tâches est énumérée dans le temps.
- Comme l'ordonnanceur round-robin attribue des tranches de temps égales à chaque tâche, la tâche la plus courte (**tâche 1**) a exécuté près de six itérations de sa boucle, tandis que la tâche dont la boucle est la plus lente (**tâche 2**) n'a exécuté que la première itération.
- **La tâche 3** a exécuté trois itérations de sa boucle.

Ordonnancement à la ronde (4/5)

- Une distinction extrêmement importante entre une super boucle exécutant les mêmes fonctions et une routine d'ordonnancement round-robin les exécutant est la suivante :
 - La tâche 3 a terminé sa boucle modérément serrée avant la tâche 2.
- Lorsque la super boucle exécutait les fonctions en série, la fonction 3 n'aurait même pas commencé avant que la fonction 2 ne soit terminée.
- Ainsi, bien que l'ordonnanceur ne nous fournisse pas un véritable parallélisme, chaque tâche reçoit sa juste part de cycles CPU.
- Ainsi, avec ce schéma d'ordonnancement, si une tâche a une boucle plus courte, elle s'exécutera plus souvent qu'une tâche ayant une boucle plus longue.

Ordonnancement à la ronde (5/5)

- Tous ces changements ont un (léger) coût : l'ordonnanceur doit être invoqué chaque fois qu'il y a un changement de contexte.
- Dans cet exemple, les tâches n'appellent pas explicitement l'ordonnanceur pour s'exécuter.
- Dans le cas de FreeRTOS fonctionnant sur un ARM Cortex-M, l'ordonnanceur sera appelé à partir de l'interruption SysTick (plus de détails peuvent être trouvés dans le chapitre 7, l'ordonnanceur de FreeRTOS).
- Des efforts considérables sont déployés pour s'assurer que le noyau de l'ordonnanceur est extrêmement efficace et que son exécution prend le moins de temps possible.
- Il n'en reste pas moins qu'il s'exécutera à un moment ou à un autre et qu'il consommera des cycles de l'unité centrale.
- Sur la plupart des systèmes, la petite quantité de frais généraux n'est généralement pas remarquée (ou significative), mais elle peut devenir un problème dans certains systèmes.

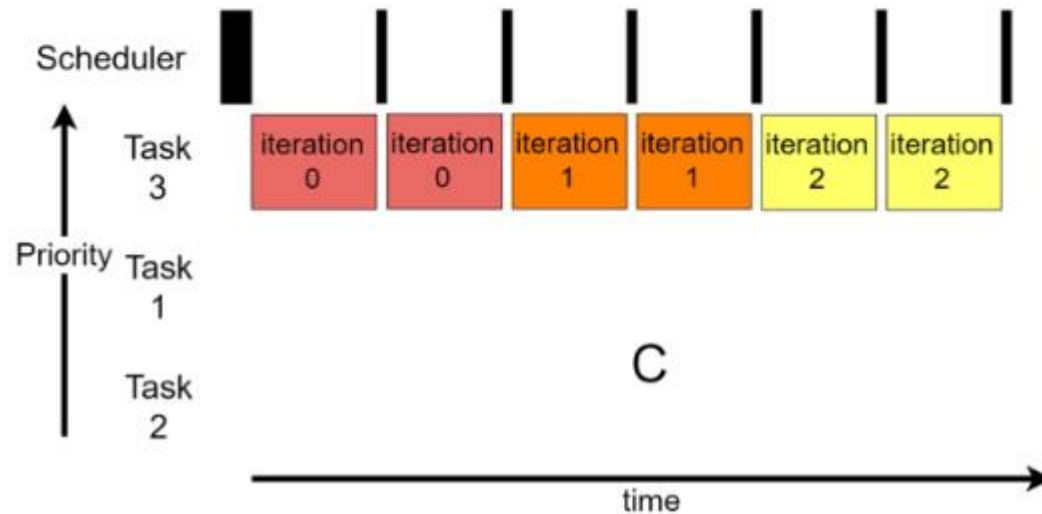
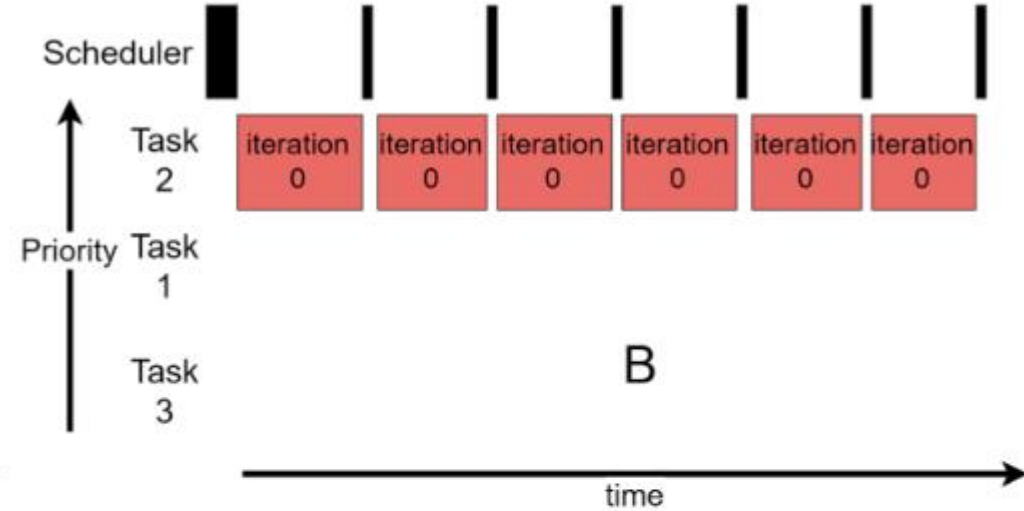
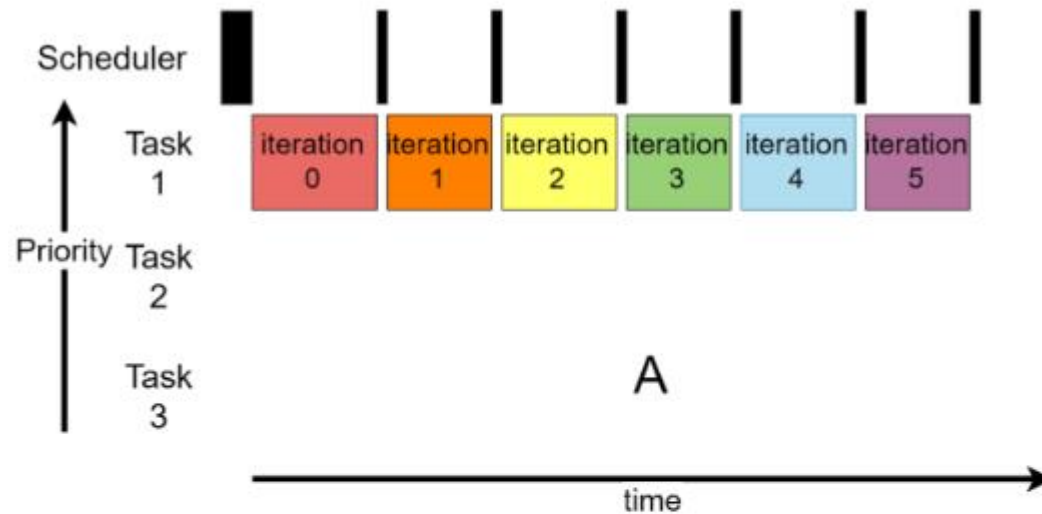
Programmation basée sur la préemption (1/6)

- L'ordonnancement préemptif fournit un mécanisme garantissant que le système exécute toujours la tâche la plus importante.
- Un algorithme d'ordonnancement préemptif donne la priorité à la tâche la plus importante, indépendamment de ce qui se passe ailleurs dans le système
 - à l'exception des interruptions, puisqu'elles se produisent sous l'ordonnanceur et ont toujours une priorité plus élevée.
- Examinons les trois mêmes tâches.
- Ces trois tâches ont la même fonctionnalité :
 - une simple boucle *while* qui incrémente sans fin une variable volatile.

Programmation basée sur la préemption (2/6)

- Envisagez maintenant les trois scénarios suivants pour déterminer laquelle des trois tâches obtiendra le contexte.
- Le diagramme suivant présente les mêmes tâches que celles présentées précédemment avec l'ordonnancement round-robin.
- Chacune des trois tâches a plus de travail qu'il n'en faut, ce qui empêchera la tâche de sortir de son contexte (voir diapositive suivante).

Basic Task Setup (Preemptive Scheduler)



- Preemptive scheduler scheduling 3 tasks that always have work to perform (non blocking)
- each task takes a different amount of time per loop iteration
- 3 different priority schemes (A, B, C)

Programmation basée sur la préemption (3/6)

- Que se passe-t-il donc lorsque trois tâches différentes sont définies avec trois séries de priorités différentes (A, B et C) ?
- **A (en haut à gauche) : La tâche 1** a la priorité la plus élevée dans le système
 - el reçoit tout le temps du processeur !
 - Quel que soit le nombre d'itérations effectuées par la tâche 1, s'il s'agit de la tâche la plus prioritaire du système et qu'elle a du travail à faire (sans attendre quoi que ce soit d'autre dans le système), elle se verra attribuer un contexte et sera exécutée.

Programmation basée sur la préemption (4/6)

- **B (en haut à droite) :** La tâche 2 est la tâche la plus prioritaire du système.
 - Étant donné qu'elle a plus qu'assez de travail à faire et qu'elle n'a pas besoin d'attendre quoi que ce soit d'autre dans le système, la tâche 2 sera mise en contexte.
 - La tâche 2 étant configurée comme la plus prioritaire du système, elle s'exécutera jusqu'à ce qu'elle doive attendre quelque chose d'autre dans le système.
- **C (en bas à gauche) :** La tâche 3 est configurée comme la tâche la plus prioritaire du système. Aucune autre tâche ne s'exécute parce qu'elle est moins prioritaire.

Programmation basée sur la préemption (5/6)

- Il est évident que si vous concevez un système nécessitant l'exécution de plusieurs tâches en parallèle, un ordonnanceur préemptif ne serait pas d'une grande utilité si toutes les tâches du système nécessitaient 100 % du temps de l'unité centrale et n'avaient pas besoin d'attendre quoi que ce soit.
- Cette configuration ne serait pas non plus une bonne conception pour un système en temps réel puisqu'elle était complètement surchargée (et ignorait deux des trois fonctions principales que le système était censé remplir) !
- Cette situation est appelée "famine des tâches", car seule la tâche la plus prioritaire du système obtient du temps de processeur et les autres tâches sont privées de temps de processeur.

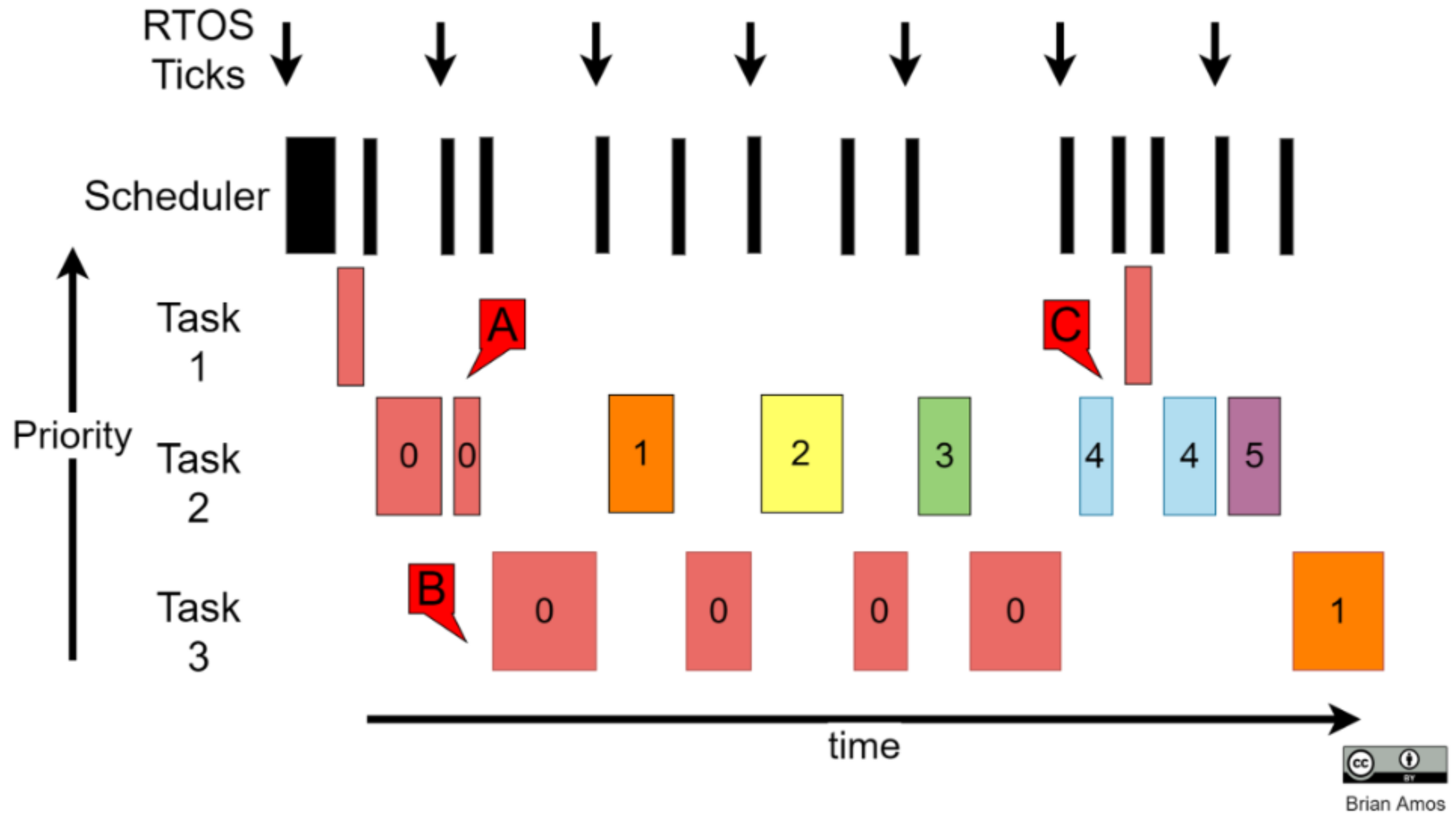
Programmation basée sur la préemption (6/6)

- Un autre détail qui mérite d'être souligné est que l'ordonnanceur continue de fonctionner à des intervalles prédéterminés.
- Peu importe ce qui se passe dans le système, l'ordonnanceur s'exécutera avec diligence à son rythme prédéterminé.
- Il existe une exception à cette règle.
- FreeRTOS dispose d'un mode d'ordonnancement sans tic-tac conçu pour les appareils à très faible consommation d'énergie, qui empêche l'ordonnancement de fonctionner aux mêmes intervalles prédéterminés.

Ordonnanceur préemptif réaliste (1/3)

- La diapositive suivante présente un cas d'utilisation plus réaliste où un ordonnanceur préemptif est utilisé.
- Dans ce cas, la **tâche 1** est la tâche la plus prioritaire du système (il se trouve également que son exécution se termine très rapidement).
 - la seule fois où la **tâche 1** se voit retirer son contexte, c'est lorsque l'ordonnanceur doit s'exécuter ; sinon, elle conservera son contexte jusqu'à ce qu'elle n'ait plus de travail supplémentaire à effectuer.
- La **tâche 2** est la priorité suivante : vous remarquerez également que cette tâche est configurée pour s'exécuter une fois par tic du l'ordonnanceur RTOS (indiqué par les flèches vers le bas).
- La **tâche 3** est la tâche la moins prioritaire du système : elle n'est contextualisée que lorsqu'il n'y a rien d'autre d'intéressant à faire dans le système.
- Trois points principaux méritent d'être examinés dans ce diagramme :

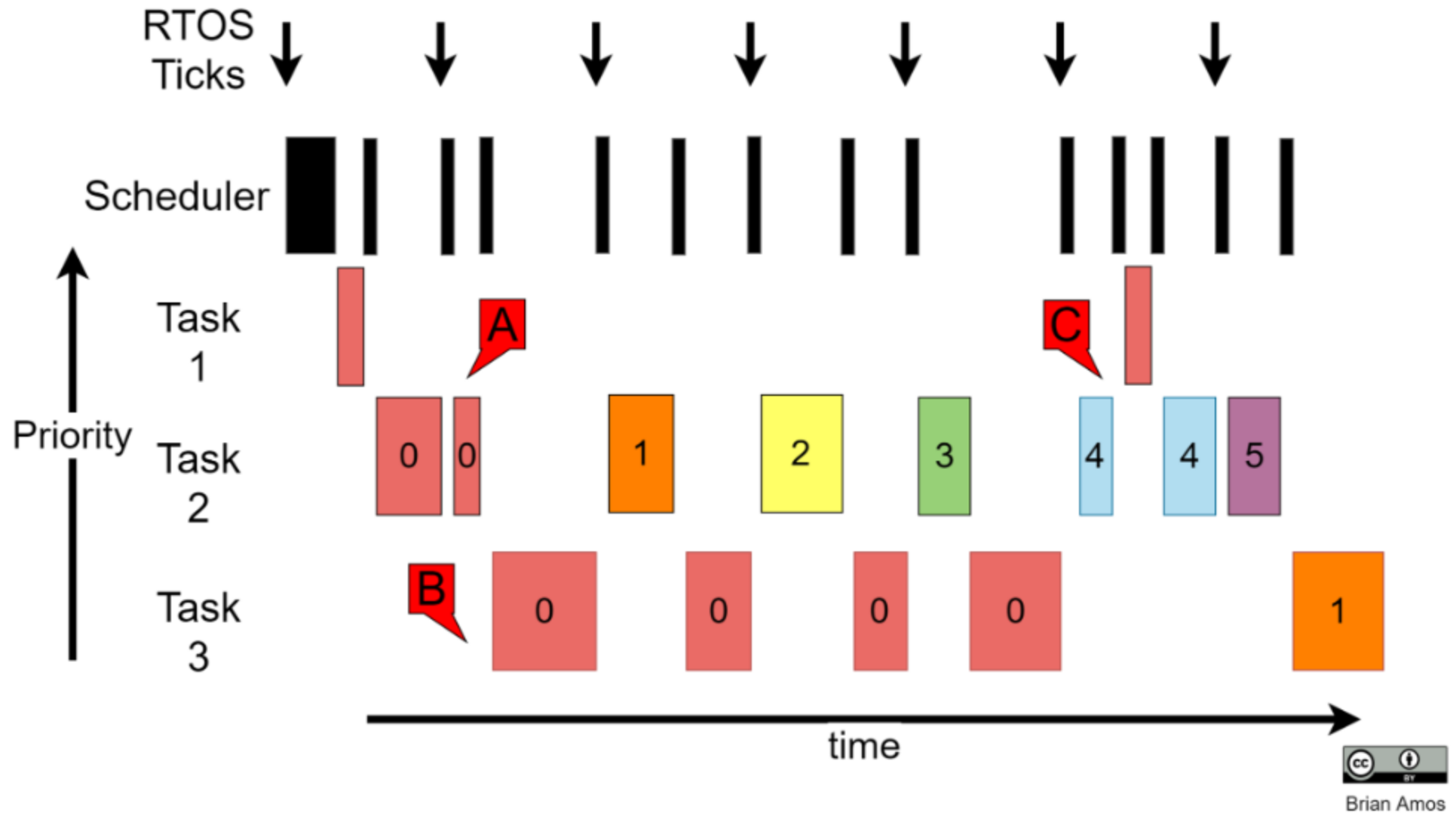
Realistic Task Setup (Preemptive Scheduler)



Ordonnanceur préemptif réaliste (2/3)

- **R** : La tâche 2 a un contexte. Même si elle est interrompue par l'ordonnanceur, elle retrouve immédiatement son contexte après l'exécution du l'ordonnanceur (parce qu'elle a encore du travail à effectuer).
- **B** : La tâche 2 a terminé son travail pour l'itération 0.
- L'ordonnanceur s'est exécuté et a déterminé que (puisque aucune autre tâche du système ne doit s'exécuter) la tâche 3 pouvait disposer d'un temps processeur.

Realistic Task Setup (Preemptive Scheduler)



Ordonnanceur préemptif réaliste (3/3)

- **C** : La tâche 2 a commencé à exécuter l'itération 4, mais la tâche 1 a maintenant du travail à faire - même si la tâche 2 n'a pas terminé le travail pour cette itération.
- La tâche 1 est immédiatement activée par l'ordonnanceur pour effectuer son travail prioritaire. Une fois que la tâche 1 a terminé ce qu'elle avait à faire, la tâche 2 est réactivée pour terminer l'itération 4.
- Cette fois, l'itération se poursuit jusqu'au prochain tic-tac et la tâche 2 s'exécute à nouveau (itération 5).
- Une fois l'itération 5 de la tâche 2 terminée, il n'y a pas de travail plus prioritaire à effectuer, de sorte que la tâche la moins prioritaire du système (la tâche 3) s'exécute à nouveau.
- Il semble que la tâche 3 ait finalement terminé l'itération 0, elle passe donc à l'itération 1 et poursuit son chemin.

RTOS Ticks

Scheduler

Task 1

Task 2

Task 3

Priority

time

CC BY

Brian Amos

Tâches RTOS contre super boucles: avantages et inconvénients (1/2)

- Les super boucles sont idéales pour les systèmes simples dont les responsabilités sont limitées.
- Si un système est suffisamment simple, il peut fournir une gigue très faible en réponse à un événement, mais seulement si la boucle est suffisamment serrée.
- Au fur et à mesure qu'un système devient plus complexe et acquiert plus de responsabilités, les taux d'interrogation diminuent.
- Cette diminution du taux d'interrogation entraîne une gigue beaucoup plus importante en réponse aux événements. Des interruptions peuvent être introduites dans le système pour lutter contre l'augmentation de la gigue.
- Plus, un système basé sur une super boucle devient complexe, plus il devient difficile de suivre et de garantir la réactivité aux événements.

Tâches RTOS contre super boucles: avantages et inconvénients (2/2)

- Un RTOS devient très utile pour les systèmes plus complexes qui ont non seulement des tâches qui prennent du temps, mais qui nécessitent également une bonne réactivité aux événements externes.
- Avec un RTOS, l'augmentation de la complexité du système, de la ROM, de la RAM et du temps d'installation initial est la contrepartie d'un système plus facile à comprendre, qui peut plus facilement garantir la réactivité aux événements externes en temps voulu.

Résumé

- Nous avons abordé dans ce chapitre un certain nombre de concepts liés aux super boucles et aux tâches.
- À ce stade, vous devriez avoir une bonne compréhension de la façon dont les super boucles peuvent être combinées avec des interruptions et des DMA pour fournir un traitement parallèle permettant à un système de rester réactif, sans utiliser de RTOS.
- Nous avons présenté les architectures basées sur les tâches à un niveau théorique et les deux principaux types d'ordonnancement que vous rencontrerez en utilisant FreeRTOS (round-robin et preemptive).
- Vous avez également eu un bref aperçu de la manière dont un ordonnanceur préemptif planifie des tâches de différentes priorités.
- Tous ces concepts sont importants à saisir, aussi n'hésitez pas à vous référer à ces exemples simplistes lorsque nous aborderons des sujets plus avancés.