Une approche synchrone à la conception de systèmes embarqués temps réel

Dumitru Potop-Butucaru

dumitru.potop@inria.fr

cours EPITA, 2023, 5ème séance

Ce que nous avons déjà fait

- Système embarqué, système réactif
  - Spécification fonctionnelle et non-fonctionnelle
- Langage réactif Lustre/Scade/Heptagon
  - Spécification fonctionnelle
    - Exemple GNC
  - Ordonnancement temps réel
    - Hors ligne – MIF/MAF
    - En ligne – préemptif à base de priorités
      - RM, EDF…

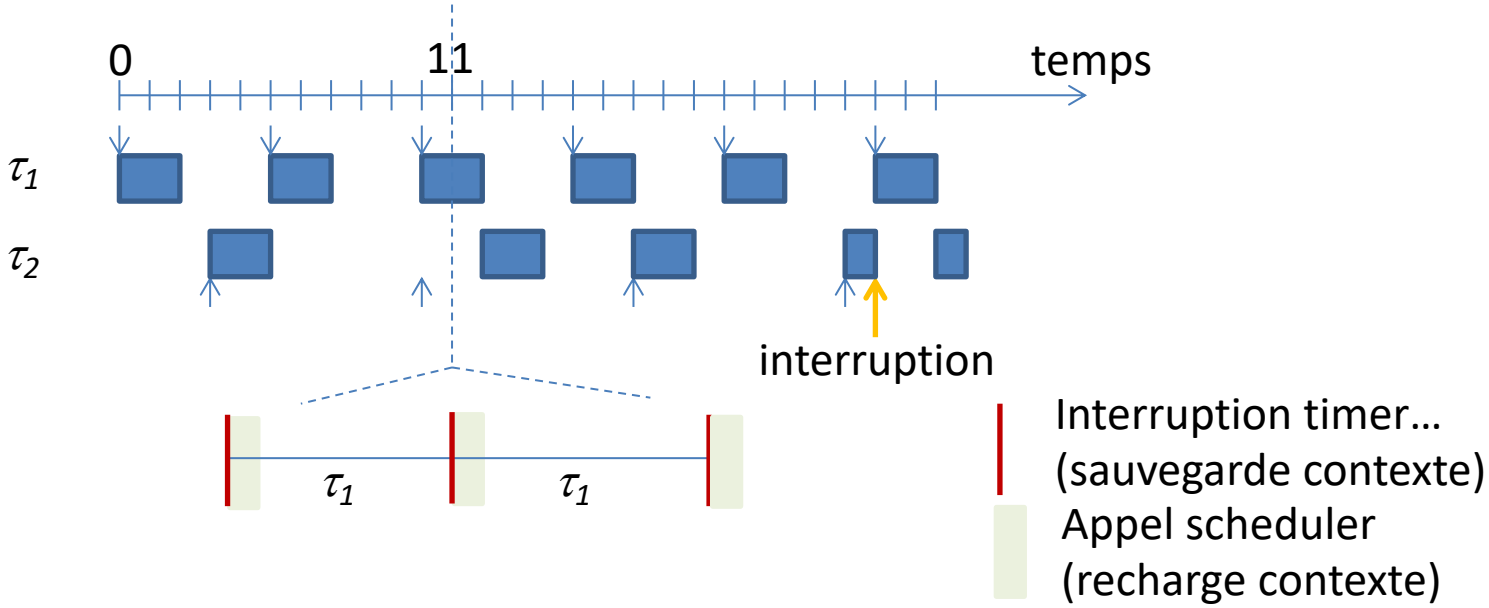
Contenu de ce cours

- Organisation d'un système (temps réel)
  - Comment implanter MIF/MAF, RM, EDF ?
- Introduction au multi-coeur

- Préparation du TP
  - Début des activités sur carte Raspberry Pi

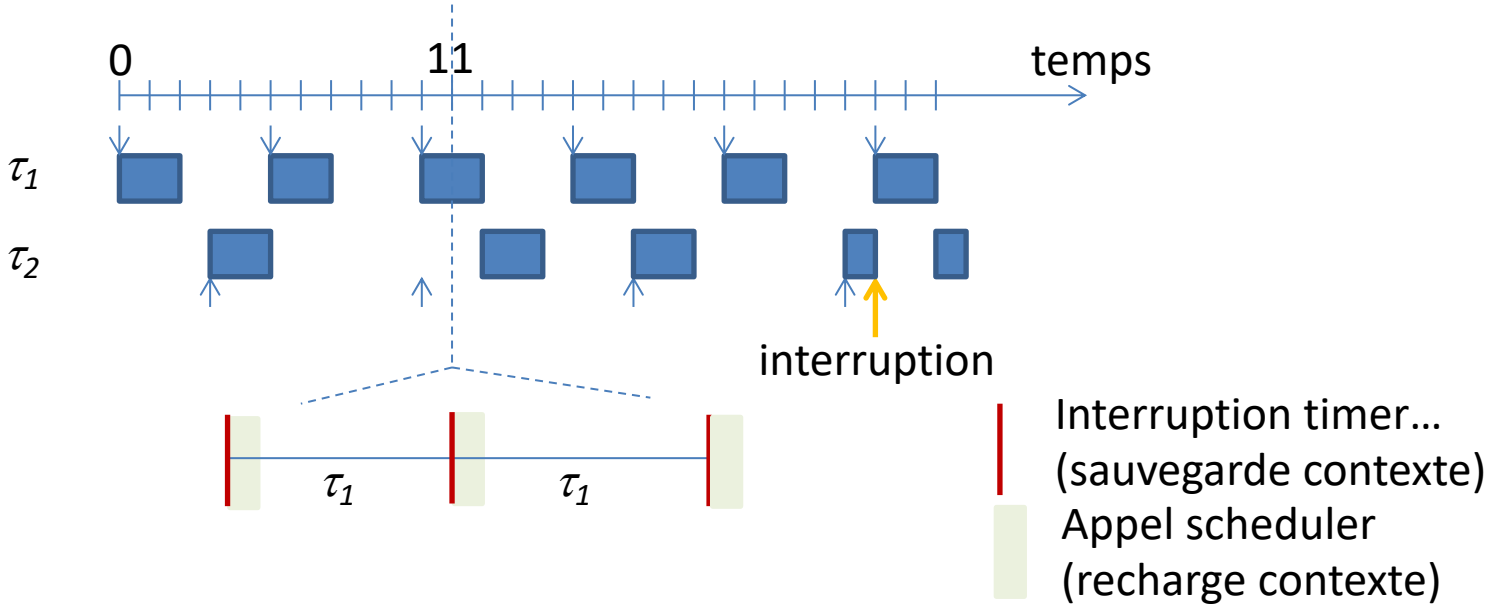
Ordonnanceur

- Ordonnanceur



Ordonnanceur

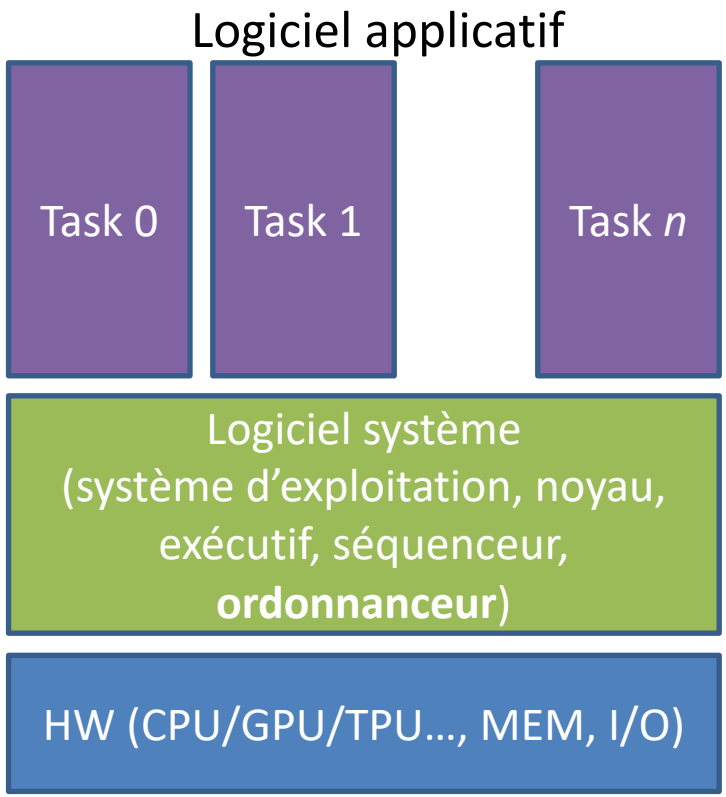
- Ordonnanceur



- *Mais qui fait quoi, au juste ?*

Organisation d'un système

- Trois principaux composants
  - HW
  - Logiciel système
    - Lien HW<->Applicatif
    - Lancement et gestion des tâches
    - Gestion des ressources matérielles
  - Logiciel applicatif



Organisation d'un système

- Interactions
  - HW->Système
    - Interruptions (e.g. IRQs – I/O, erreurs, timers, DMAs…)
    - Registres d'état, mémoire
  - Système->HW: registres de contrôle
  - Système->Applicatif
    - Passage de contrôle (start, resume), valeurs de retour
  - Applicatif->Système
    - Services système (e.g. usleep, gettimeofday…)
  - HW->applicatif
    - Certains registres d'état (e.g. arithmétique)
  - Applicatif -> matériel
    - Exécution
    - Contrôle seulement dans des cas particuliers (e.g. ressources non-partagées…)

Organisation d'un système

- Interactions typiques :
  - HW:
    - Une trame réseau (ou un timer ou la fin d'un transfert DMA ou une erreur d'accès mémoire…) génère une IRQ qui interrompt l'exécution d'une tâche et donne le contrôle au noyau
  - Le noyau:
    - Sauvegarde le contexte de la tâche (pour permettre sa reprise plus tard).
    - Traite l'interruption matérielle
    - Appelle l'ordonnanceur pour choisir la tâche à reprendre ou à démarrer.
    - Reprend ou démarre la tâche choisie (ou passe en mode « attente » si aucune tâche ne doit être exécutée).
  - La tâche qui s'exécute peut:
    - Demander au système la date courante
    - Signaler sa terminaison auprès du système -> ordo
    - Faire une erreur dans l'utilisation du matériel
      - E.g. débordement arithmétique, accès à une zone mémoire interdite…

Organisation d'un système

- Chaque architecture matérielle offre support à ces mécanismes d'exécution
  - E.g. ARM 64 bits (AArch64)
    - Interruptions
    - Niveaux de privileges (+services systèmes correspondants)
      - User//OS kernel - Linux//Hypervisor - VirtualBox//Security monitor
    - MMU – Memory Management Unit
      - Configuration des accès mémoire
  - E.g. Système sur Puce BCM 2837
    - Timers
    - DMA (transferts mémoire rapides)…

Organisation d'un système

- Organisations typiques
  - Bare metal/séquenceur
    - Espace mémoire unique, sans isolation entre composants, une seule fonction appelée par une interruption « timer » unique, I/O réalisées par le logiciel applicatif (sampling).
    - Exemple GNC: la fonction « main » peut être exécutée sur timer
  - RTOS léger
    - Espace mémoire unique, sans isolation entre composants.
    - Plusieurs tâches, ordonnées par un ordonnanceur à base de priorités
    - Exemple : RM sur FreeRTOS
  - Système d'exploitation
    - Exploitation du système de privilèges
    - Isolation mémoire
    - Interface Applicatif/Système très développée
      - E.g. POSIX
  - Hyperviseur
    - Virtualisation des ressources
    - Partitionnement des ressources entre plusieurs OS/RTOS…
    - E.g. ARINC 653

Implantation multi-coeurs

Raspberry Pi 3

- Broadcom BCM2837B0 SoC
  - Processors
    - 64-bit quad-core ARM Cortex-A53 processor
      - ARMv8 architecture
      - AArch64 and AArch32 instruction sets
    - VideoCore IV runs at 400MHz
    - Synchronization through HW mailboxes
  - 1Go RAM
  - I/O
    - GPIO
    - USB
    - Ethernet (par USB)
    - HDMI, analog audio, caméra vidéo (par VideoCore)

Raspberry Pi 3

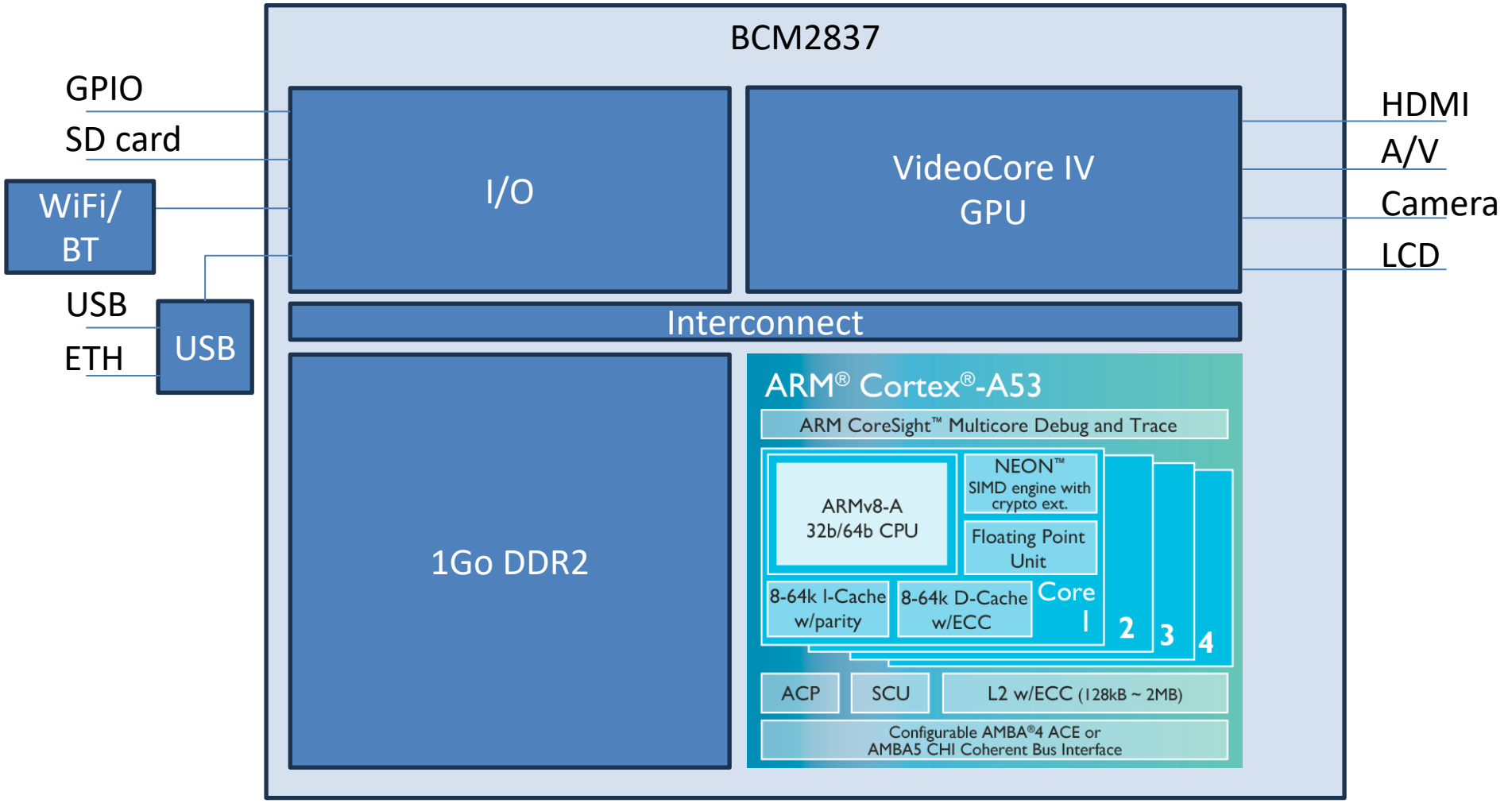
- Broadcom BCM2837B0 SoC – very well documented
  - Processors
    - 64-bit quad-core ARM Cortex-A53 processor
      - ARMv8 architecture
      - AArch64 and AArch32 instruction sets
    - VideoCore IV runs at 400MHz
    - Synchronization through HW mailboxes
  - 1Go RAM
  - I/O
    - GPIO
    - USB
    - Ethernet (par USB)
    - HDMI, analog audio, caméra vidéo (par VideoCore)

Raspberry Pi 3

- Broadcom BCM2837B0 SoC – our focus
  - Processors
    - 64-bit quad-core ARM Cortex-A53 processor
      - ARMv8 architecture
      - AArch64 and AArch32 instruction sets
    - VideoCore IV runs at 400MHz
    - Synchronization through HW mailboxes
  - 1Go RAM
  - I/O
    - GPIO
    - USB
    - Ethernet (par USB)
    - HDMI, analog audio, caméra vidéo (par VideoCore)

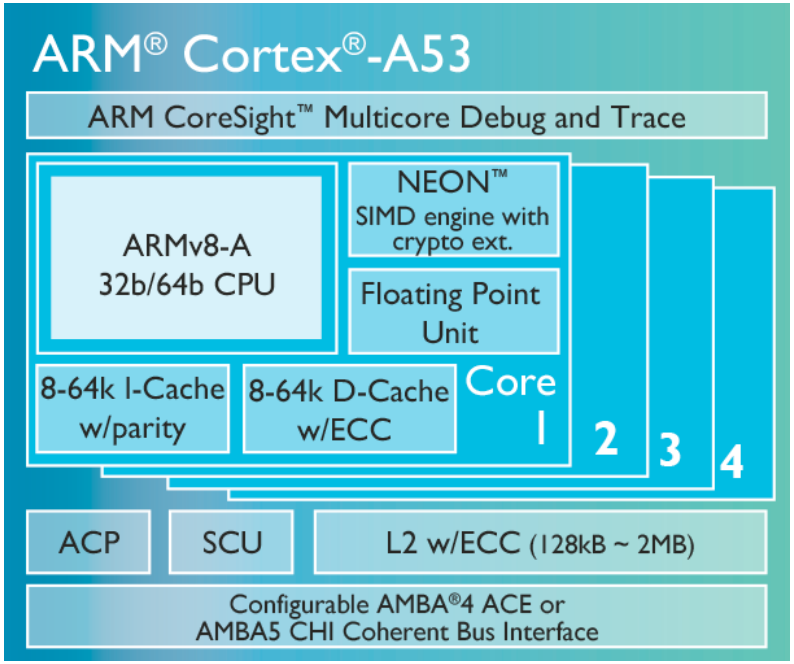
Raspberry Pi 3

- Broadcom BCM2837B0 SoC – our focus



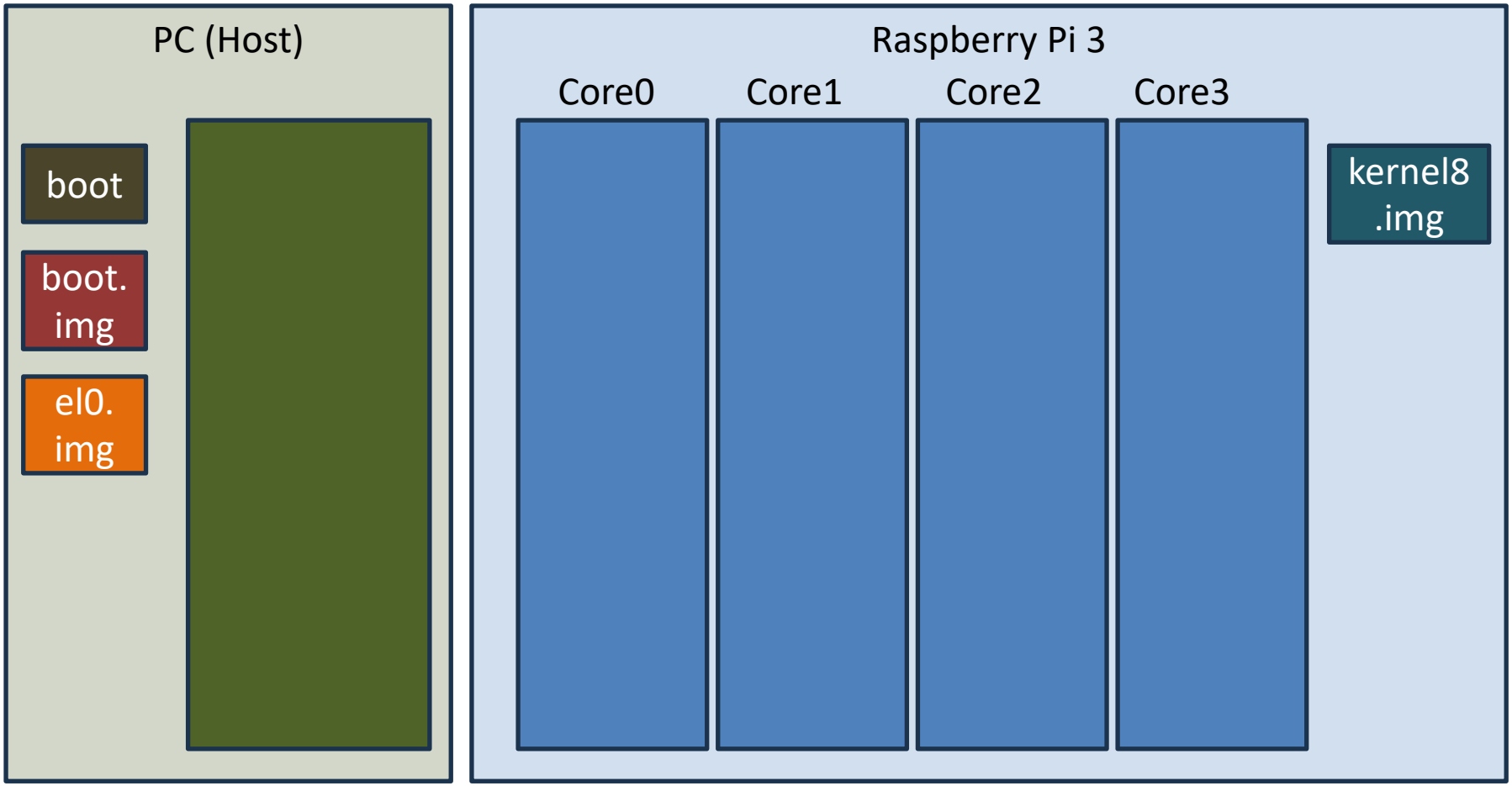
Focus sur le multi-coeurs ARM

- Quadri-coeur Cortex-A53
  - Chaque coeur
    - Fréquence 1.4GHz
      - Variable, mais fixée à 1.4GHz dans nos travaux
    - FPU, unité vectorielle NEON
    - Cache L1: 16kB+16kB
  - Cache L2 partagé
    - 512kB
  - Cohérence de caches
    - Indispensable à la synchronization efficace



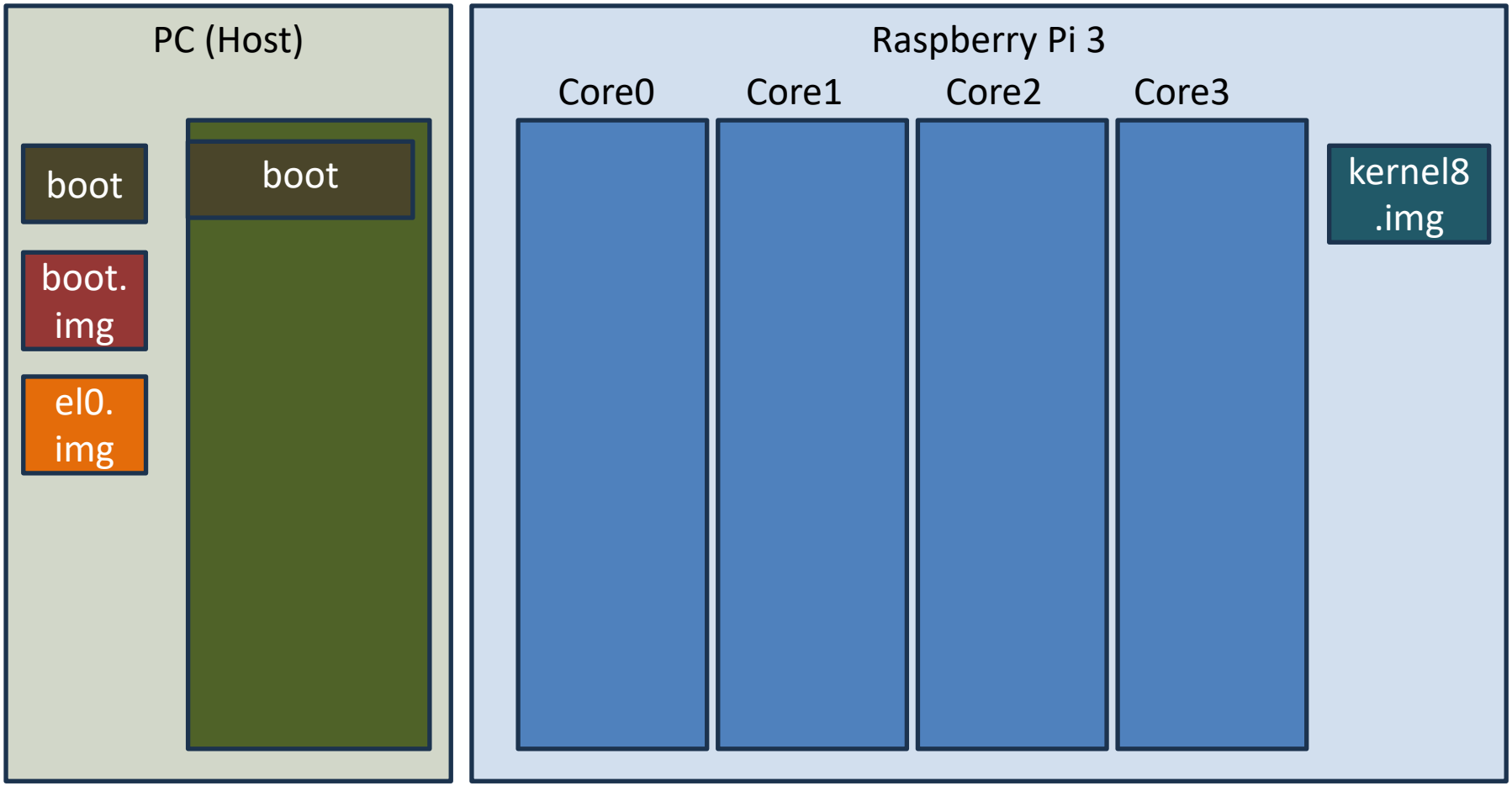
Exécuter du code

- Etat initial



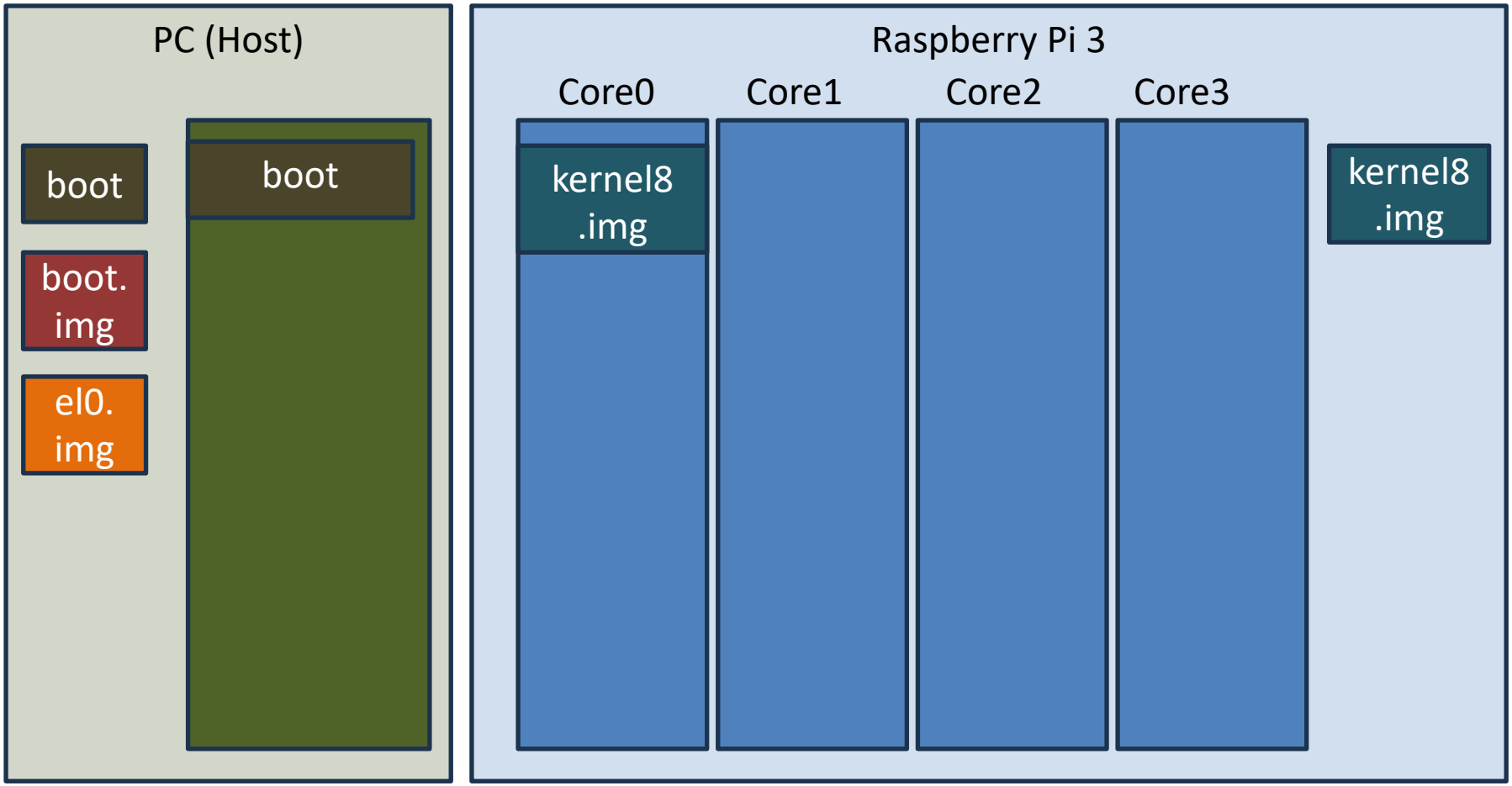
Exécuter du code

- Lancement de “boot” sur le hôte



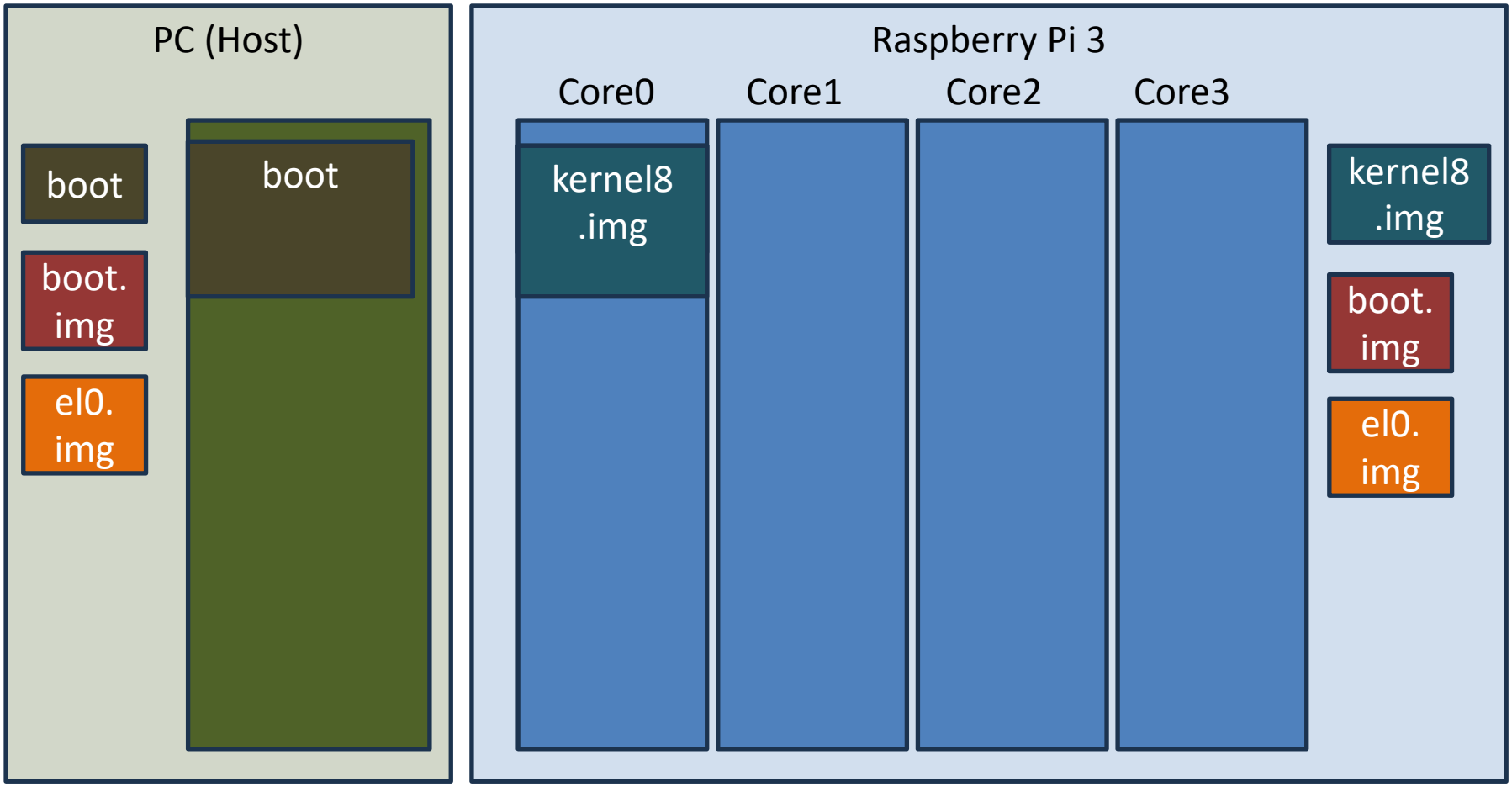
Exécuter du code

- Mise sous tension RPi3: lancement kernel8.img



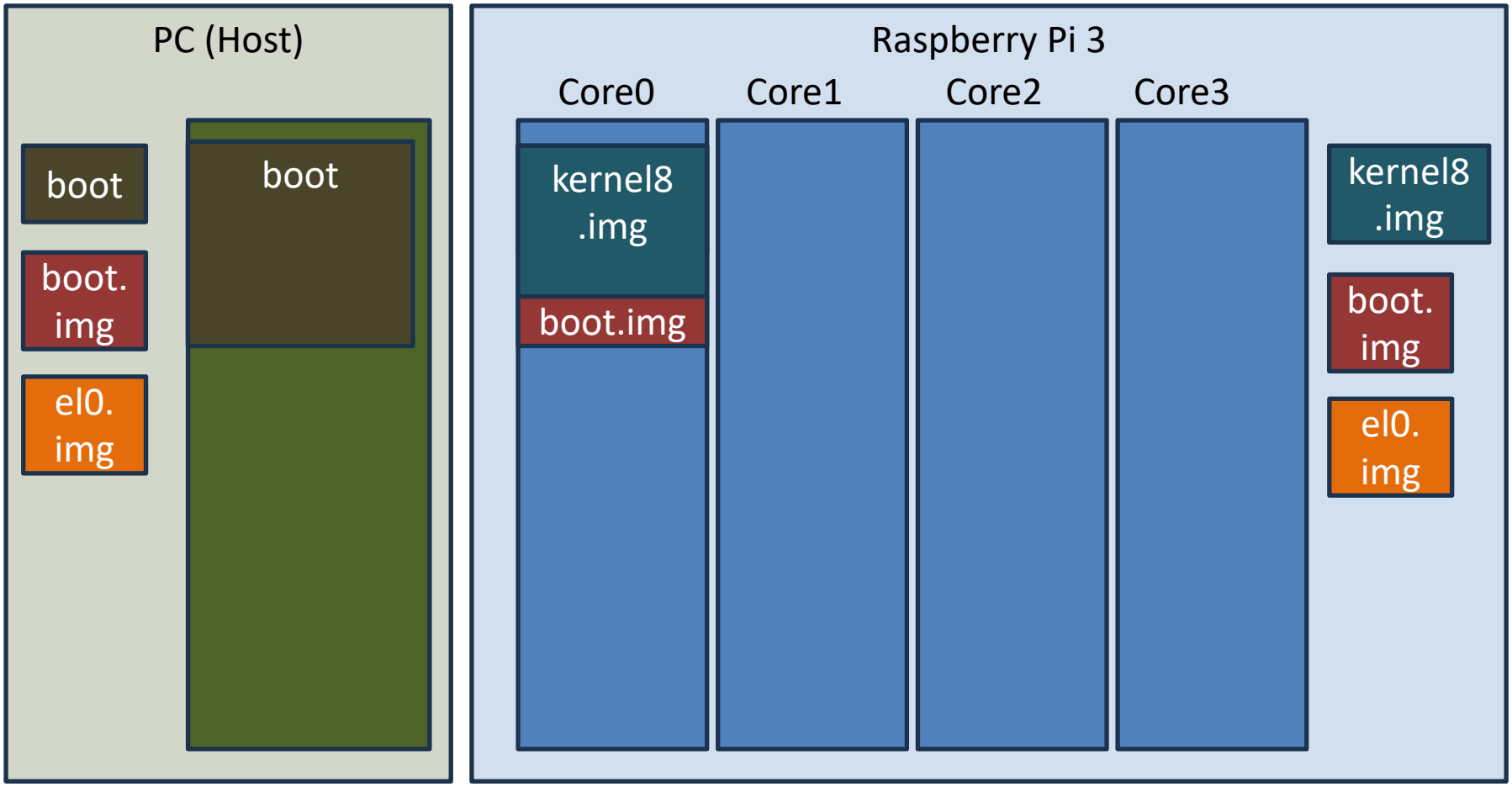
Exécuter du code

- kernel8.img: transfert de boot.img, el0.img



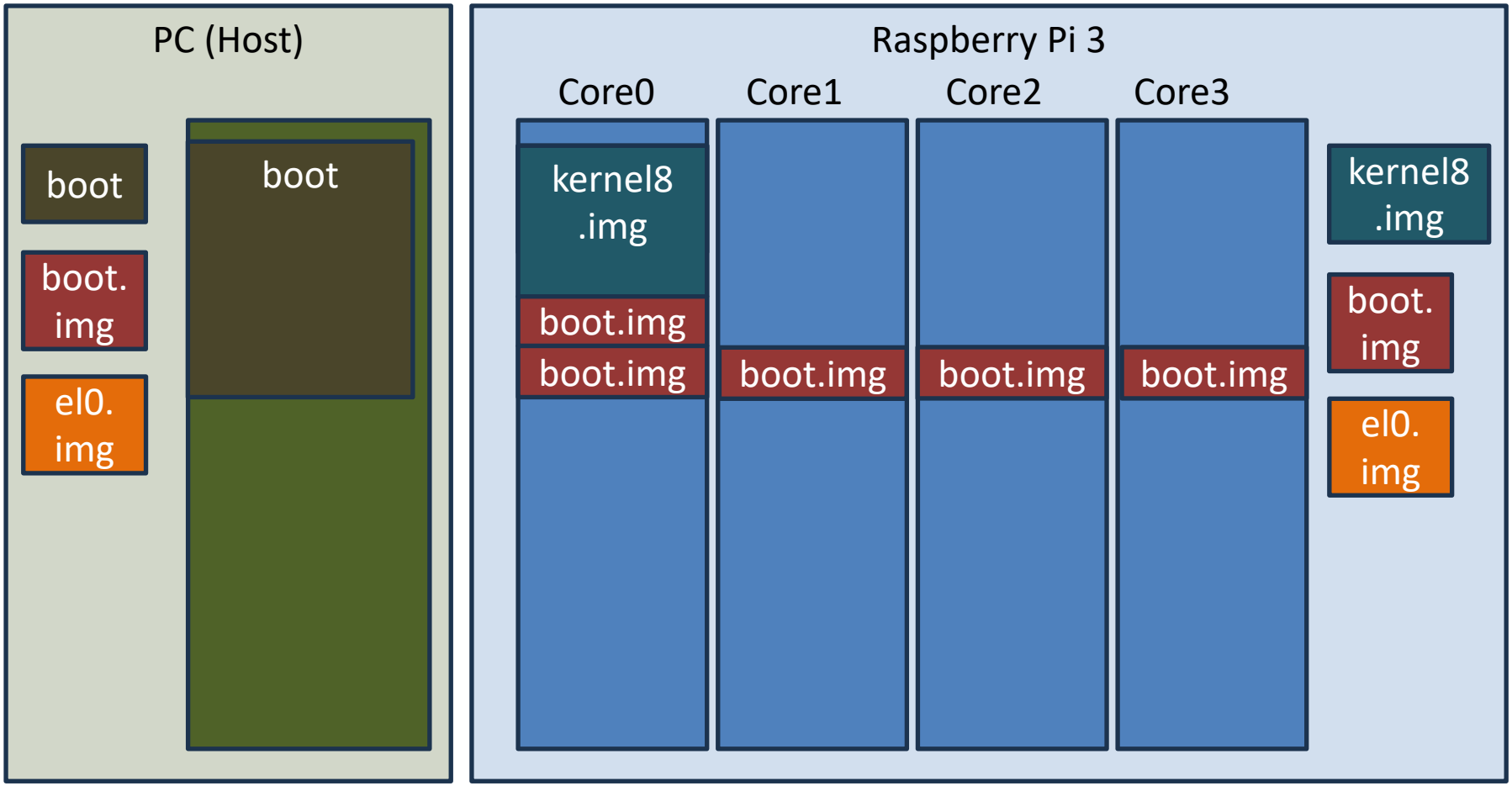
Exécuter du code

- boot.img: étape 1: configuration globale du SoC



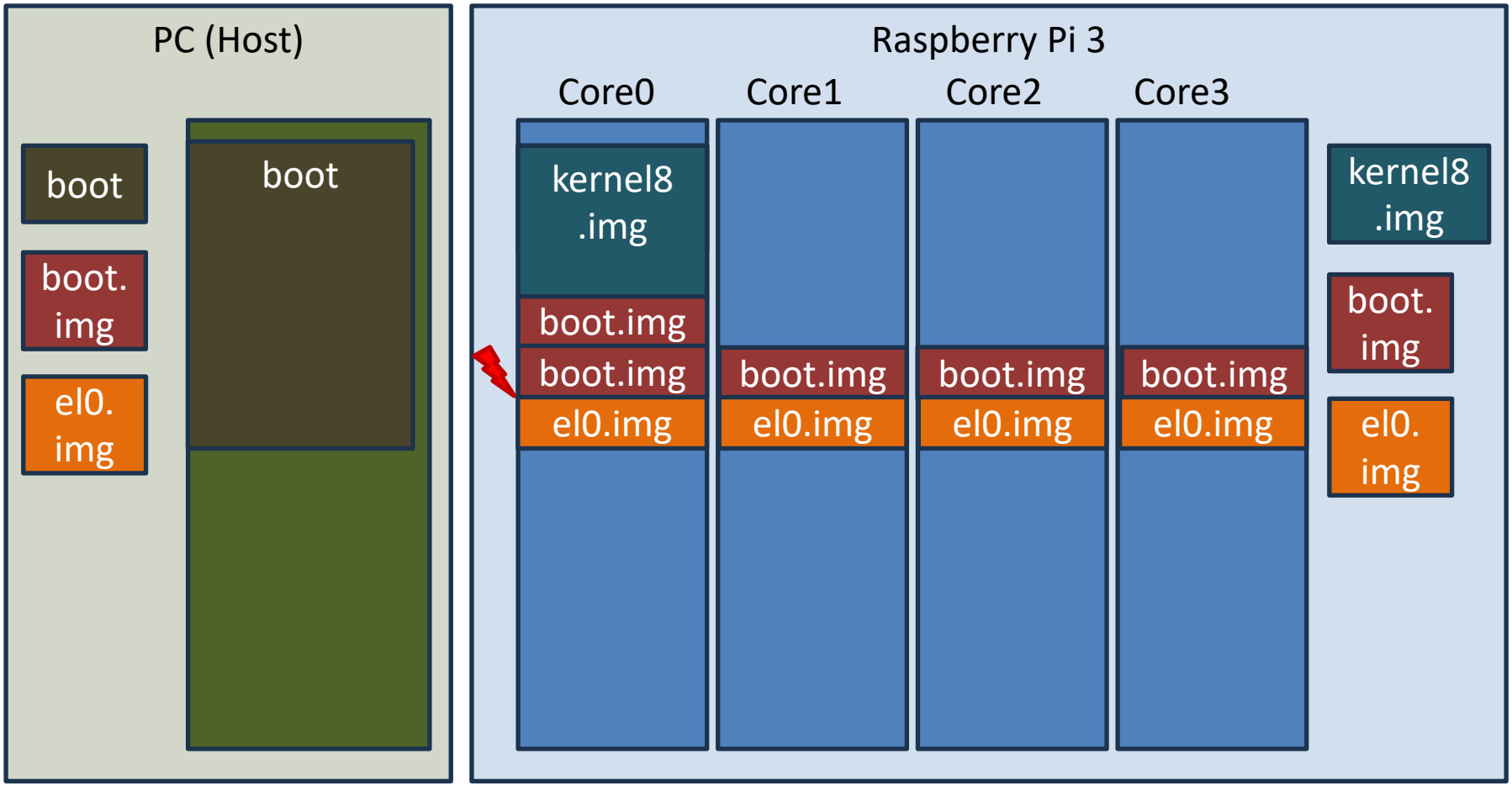
Exécuter du code

- boot.img: étape 2: configuration des coeurs



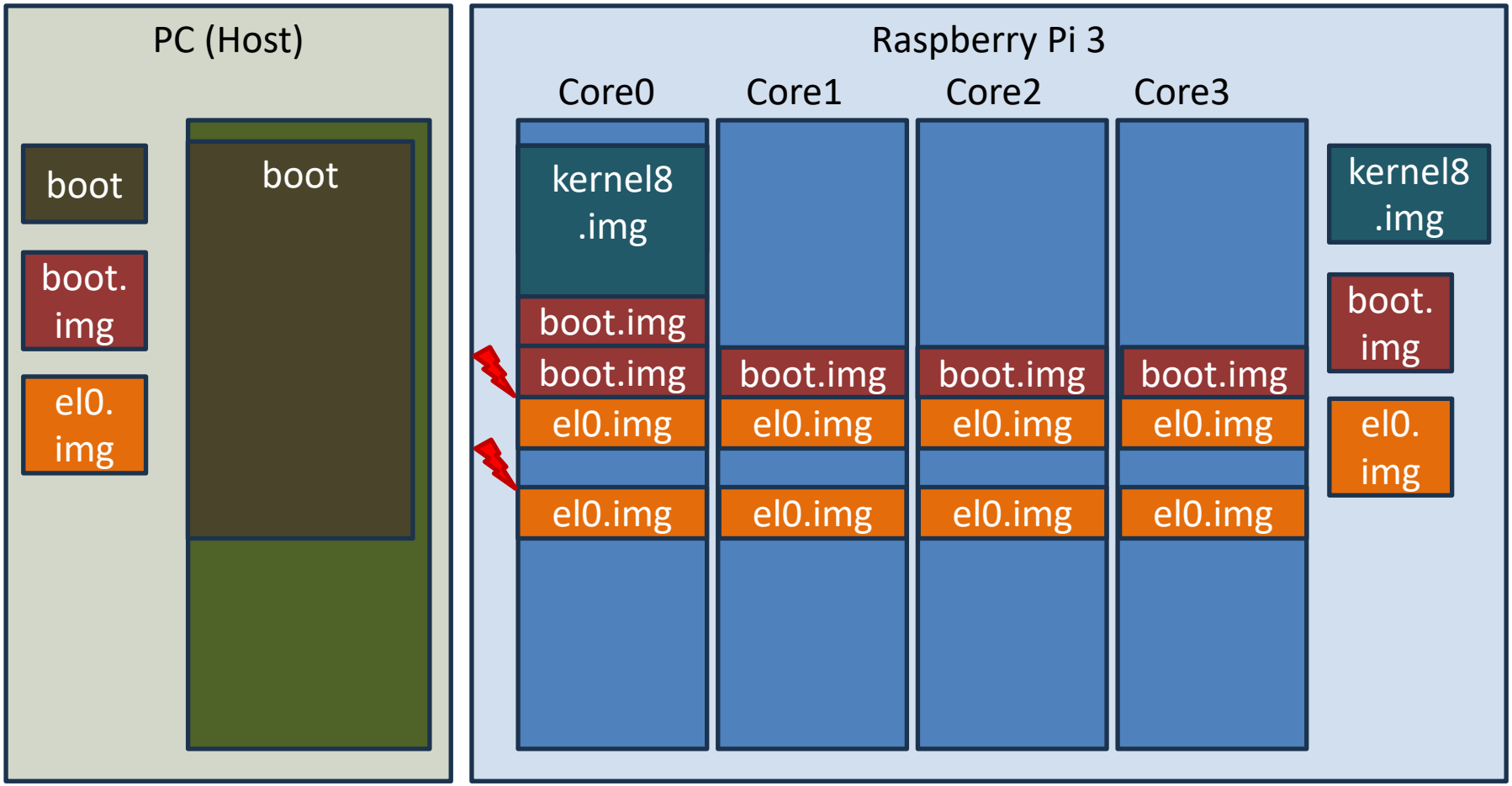
Exécuter du code cyclique

- el0.img: configuration et déclenchement timer



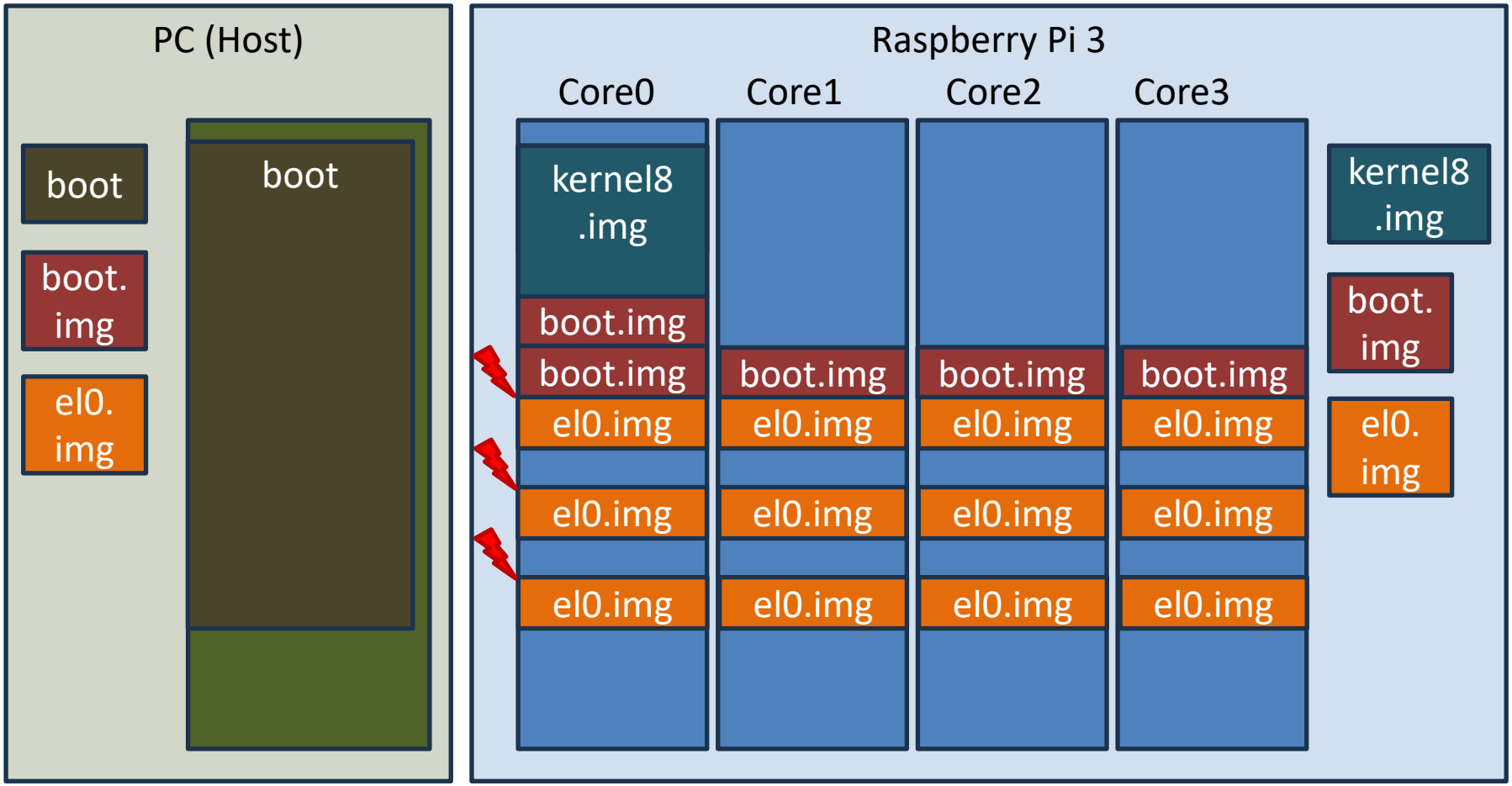
Exécuter du code cyclique

- el0.img: déclenchement timer (période = 1s)



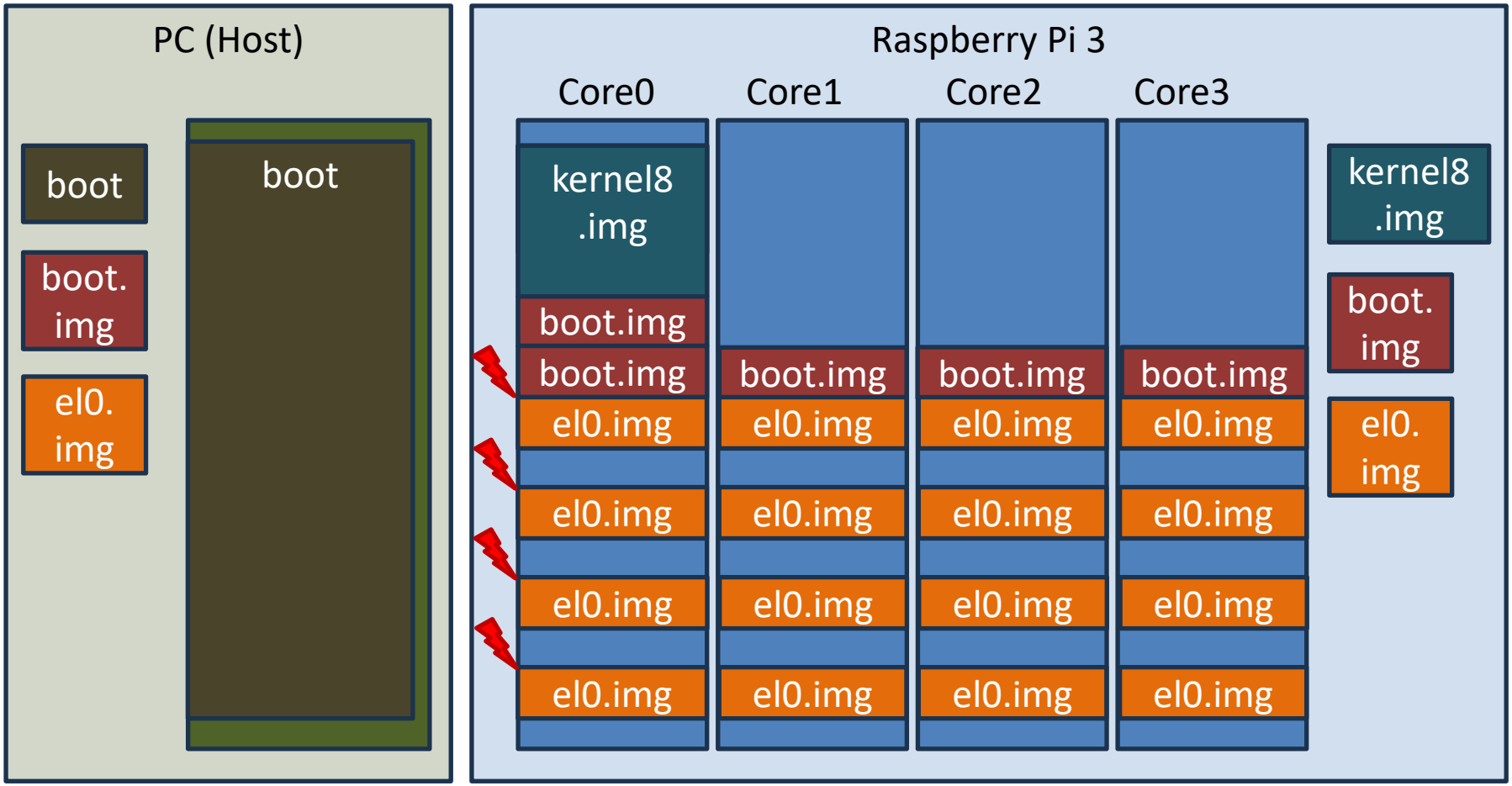
Exécuter du code cyclique

- el0.img: déclenchement timer (période = 1s)



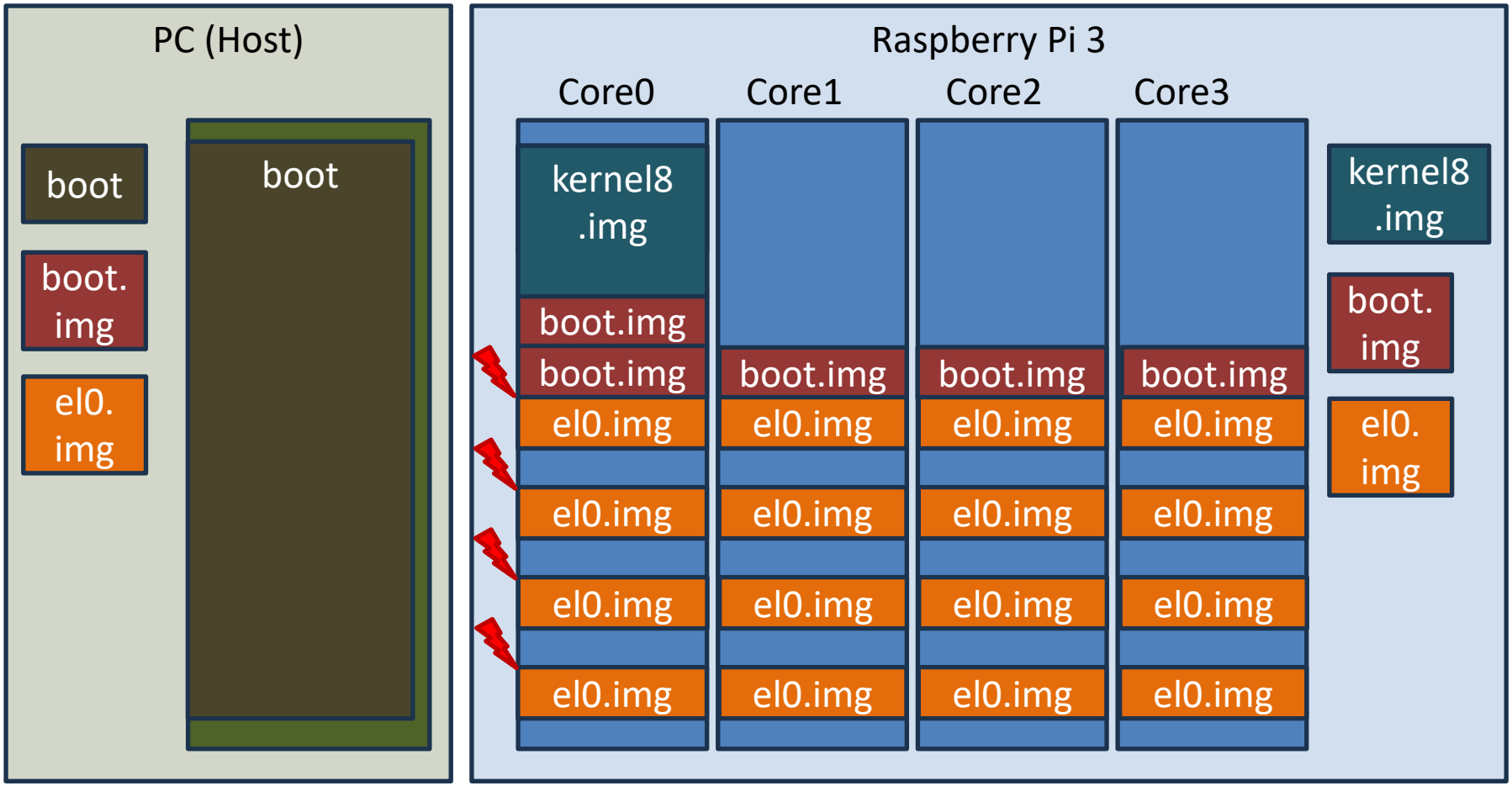
Exécuter du code

- el0.img: déclenchement timer (période = 1s)



Prenons une application

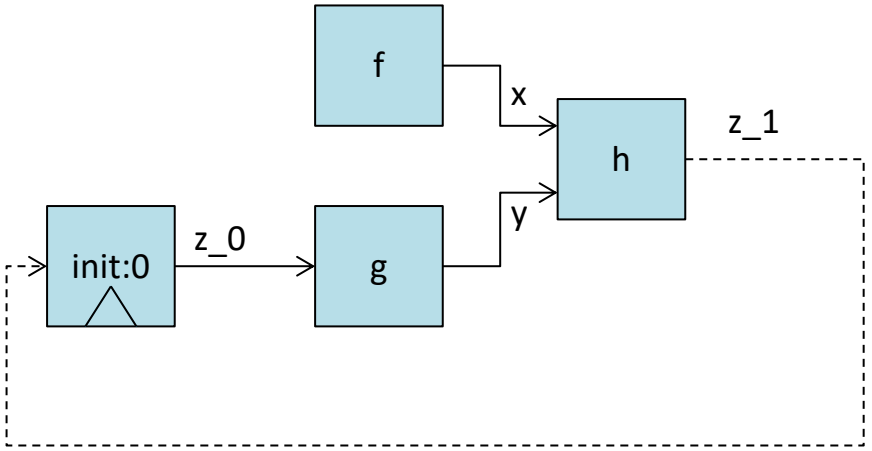
- el0.img: déclenchement timer (période = 1s)



Prenons une application Heptagon

- Cyclique, déterministe

```
open Externc (* fonctions f, g, h *)

node main () returns ()
var
  x,y:int ;
  z_0,z_1 : int at z_0 ;
let
  init<<z_0>> z_0 = 123 fby z_1 ;
  x = f() ;
  y = g(z_0) ;
  z_1 = h(x,y) ;
tel
```



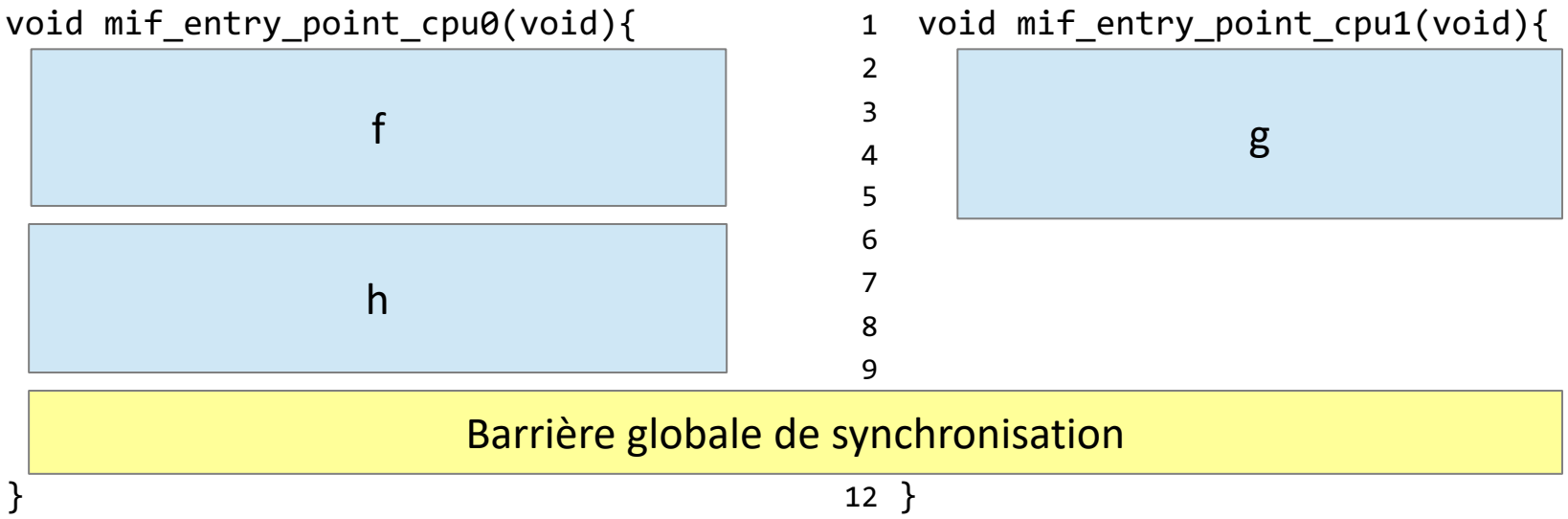
Real-time parallelization

```
void mif_entry_point_cpu0(void){
  UPDATE_CPU(loc_pc_0,0);
  g_step(z_1, &y);


  UPDATE_CPU(loc_pc_0,1000);
  WAIT_CPU(loc_pc_1,10000);
  h_step(x, y, &z_1);

  loc_pc_1 = -1 ;
  UPDATE_CPU(loc_pc_0,-1) ;
}
```

```
1  void mif_entry_point_cpu1(void){
2     UPDATE_CPU(loc_pc_1,0);
3     f_step(&x) ;
4
5
6
7
8
9
10    UPDATE_CPU(loc_pc_1,10000) ;
11
12 }
```

- Implantation bi-coeurs déterministe, data race free
  - Allocation mémoire complètement statique

Real-time parallelization



- Parallélisation des blocs flot de données
  - Ordonnancement hors ligne
    - Table d'ordonnancement
    - Réservations au pire cas

Real-time parallelization

```
void mif_entry_point_cpu0(void){
  UPDATE_CPU(loc_pc_0,0);
  g_step(z_1, &y);

  UPDATE_CPU(loc_pc_0,1000);
  WAIT_CPU(loc_pc_1,10000);
  h_step(x, y, &z_1);

  loc_pc_1 = -1 ;
  UPDATE_CPU(loc_pc_0,-1) ;
}
```

```
1  void mif_entry_point_cpu1(void){
2    UPDATE_CPU(loc_pc_1,0);
3    f_step(&x) ;
4
5
6
7
8
9
10
11     UPDATE_CPU(loc_pc_1,10000) ;
12 }
```

- Code de synchronization très portable
  - Sous-ensemble de C11
  - Déterminisme fonctionnel
  - La synchronisation facilite le respect du temps reel
    - Nombre borne d'appels par bloc flot de données

Real-time parallelization

```
void mif_entry_point_cpu0(void){
  UPDATE_CPU(loc_pc_0,0);
  g_step(z_1, &y);

  UPDATE_CPU(loc_pc_0,1000);
  WAIT_CPU(loc_pc_1,10000);
  h_step(x, y, &z_1);

  loc_pc_1 = -1 ;
  UPDATE_CPU(loc_pc_0,-1) ;
}
```

```
1  void mif_entry_point_cpu1(void){
2    UPDATE_CPU(loc_pc_1,0);
3    f_step(&x) ;
4
5
6
7
8
9
10
11     UPDATE_CPU(loc_pc_1,10000) ;
12 }
```

- Code à déclencher cycliquement
  - Contenu dans el0.img
  - Les coeurs non-utilisés exécutent une boucle infinie

Interface avec le code système

```
void mif_entry_point_cpu0(void){
  UPDATE_CPU(loc_pc_0,0);
  g_step(z_1, &y);

  UPDATE_CPU(loc_pc_0,1000);
  WAIT_CPU(loc_pc_1,10000);
  h_step(x, y, &z_1);

  loc_pc_1 = -1 ;
  UPDATE_CPU(loc_pc_0,-1) ;
}
```

```
1  void mif_entry_point_cpu1(void){
2    UPDATE_CPU(loc_pc_1,0);
3    f_step(&x) ;
4
5
6
7
8
9
10
11     UPDATE_CPU(loc_pc_1,10000) ;
12 }
```

- Fonctions mif_entry_point se trouvant sous “application/gen_t1042/threads/thread_cpuX.c”
- Code appelé sur interruption se trouvant sous “application/main.c”
  - Ce code est appelé aussi au premier cycle d’execution – initialization mémoire nécessaire
- Allocation des variables de communication (x,y,z_0,z_1) sous “application/gen_t1042/variables.[ch]”

Préparation du TP

- Document TP5.pdf
  - Mise en oeuvre du matériel et du logiciel
  - Démonstration
  - Lors du traçage, remarquez la concurrence entre les coeurs