Une approche synchrone à la conception de systèmes embarqués temps réel

Dumitru Potop-Butucaru

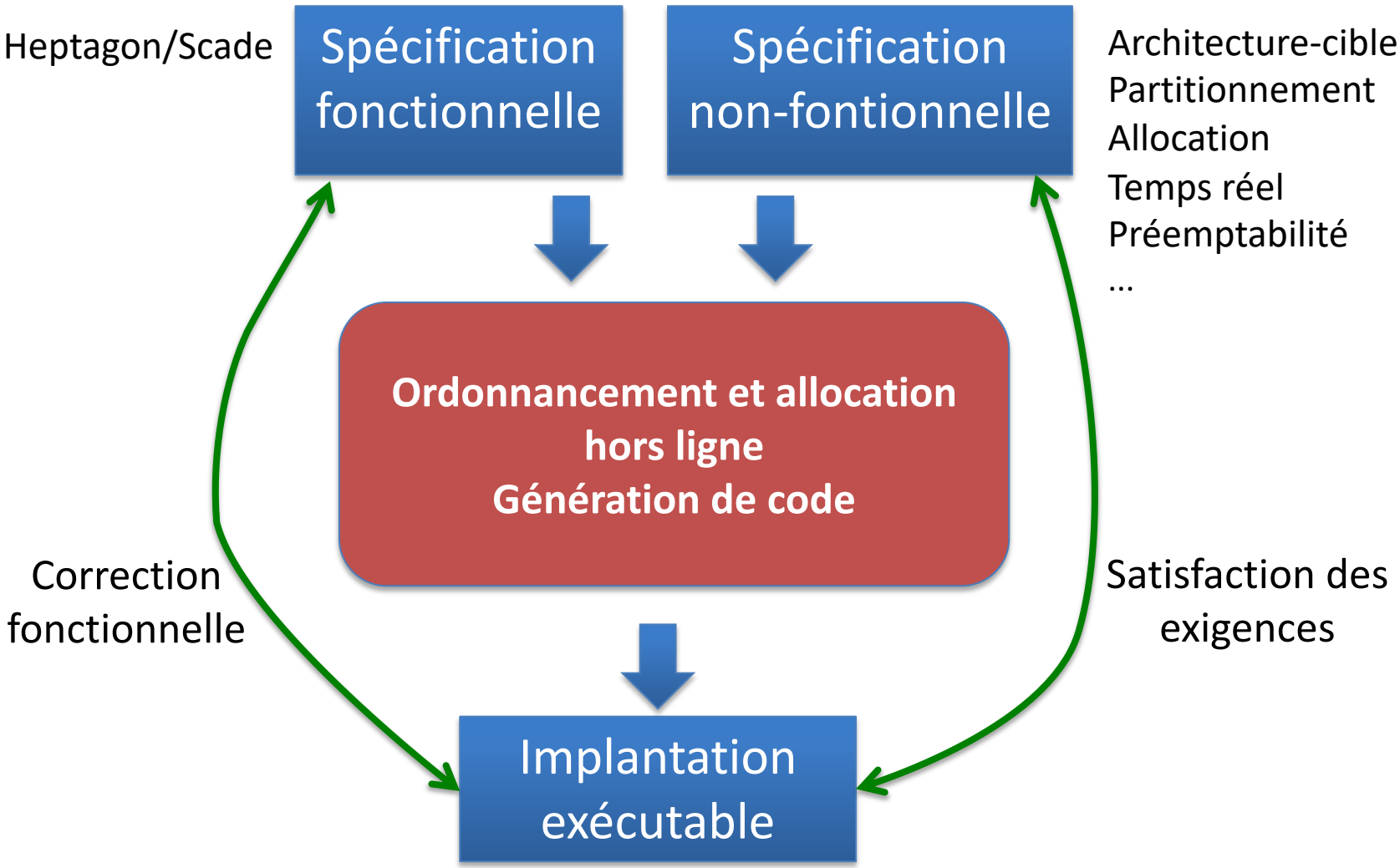
dumitru.potop@inria.fr

cours EPITA, 2022, 7ème séance

Contenu du cours

- Cours précédents
  - Programmation synchrone
  - ARINC 653 – partitionnement spatial et temporel
- Aujourd'hui
  - Cours: Faire les deux ensemble **manuellement**

Systèmes embarqués: implémentation



Construire un système manuellement

- Spécification fonctionnelle:

Construire un système manuellement

- Spécification fonctionnelle:

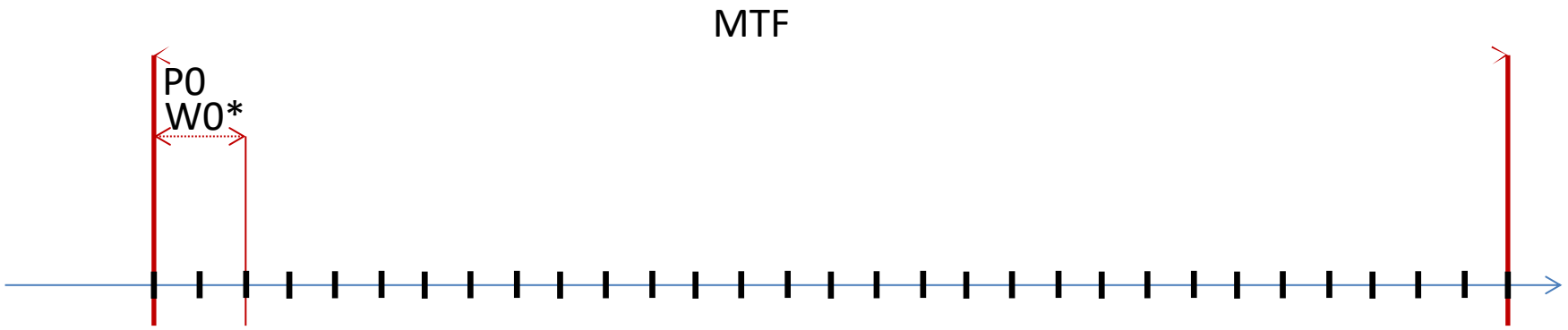
- Exigences non-fonctionnelles :
  - Traçabilité:
    - Impression par debug_print des entrées et des sorties de f et g
  - Partitionnement
    - f écrite en Heptagon, dans une partition nommée part1
    - g écrite en Heptagon, dans une partition nommée part2
    - Vous avez le choix du placement de fby
  - Période = 0xf0000 us (approx. 1sec)

Construire un système manuellement

- Spécification non-fonctionnelle:
  - HW: une SBC ARM11 de type Raspberry Pi 1
  - OS: RPi653
  - Configuration OS requise
    - Tick OS: 0x8000 us
    - La partition système demande une fenêtre de 2 ticks de longueur (0x10000)

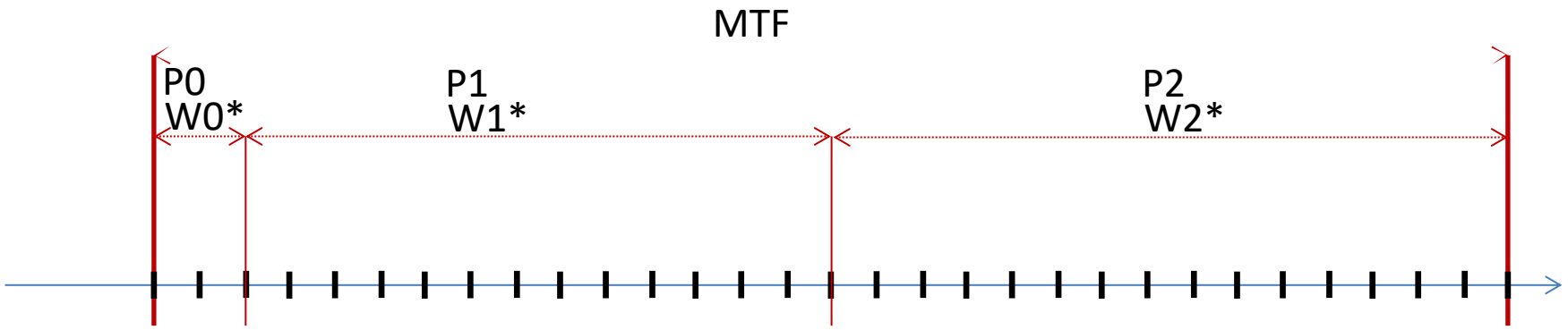
Mettre en place un MTF

- Choix de MTF = période
  - 30 ticks
  - Allouer les deux premiers ticks à une fenêtre associée à la partition système



Configuration des partitions

- Choix de MTF = période
  - 30 ticks
  - Allouer les deux premiers ticks à une fenêtre associée à la partition système
  - Le reste du MTF
    - Le diviser entre part1 et part2 – deux fenêtres de type PPS



Configuration des partitions

- Configuration:

```
tick 0x8000
mtf  0xf0000

partitions 3
0                    0xf0000
ports 0
1 part1.elf 0x100000 0xf0000
ports 2
0 out1  1
1 in1   0
2 part2.elf 0x100000 0xf0000
ports 2
0 in2   0
1 out2  1
```

```
windows 3
0       0x0      0x10000 0 1
1       0x10000  0x70000 1 1
2       0x70000  0xf0000 2 1

channels 2
0  1 0  2 0  4
1  2 1  1 1  4
```

Ne jamais faire du copier-coller !

Configuration des partitions

- Configuration des ports et des canaux

```
tick 0x8000
mtf  0xf0000

partitions 3
0                       0xf0000
ports 0
1 part1.elf 0x100000 0xf0000
ports 2
0 out1  1
1 in1   0
2 part2.elf 0x100000 0xf0000
ports 2
0 in2   0
1 out2  1
```

Port destination

Port source

```
windows 3
0       0x0      0x10000 0 1
1       0x10000  0x70000 1 1
2       0x70000  0xf0000 2 1

channels 2
0  1 0  2 0  4
1  2 1  1 1  4
```

Identifiant

Partition src

Port

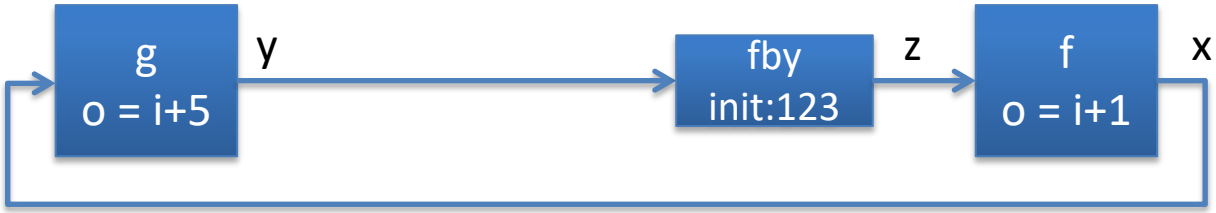
Partition dst

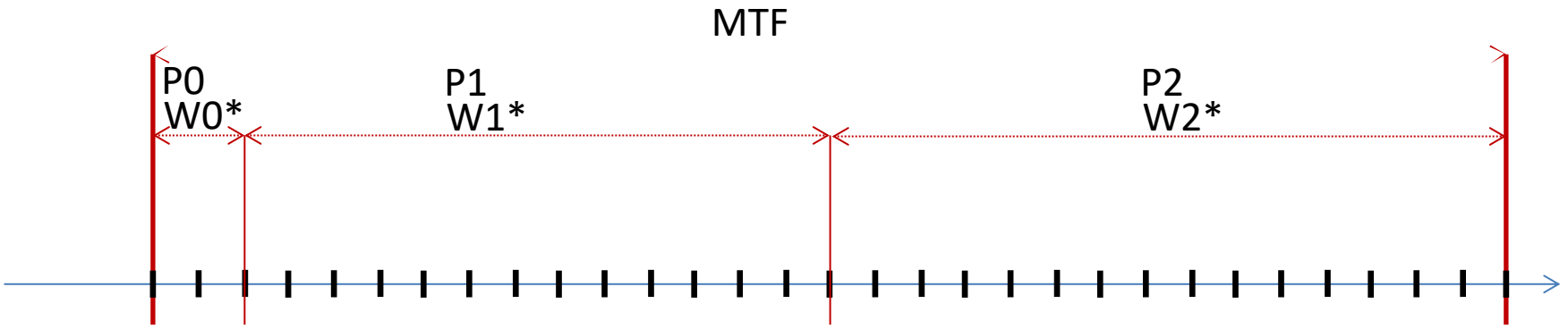
Port

Taille max msg (4 octets = int32)

Objectif d'ordonnancement

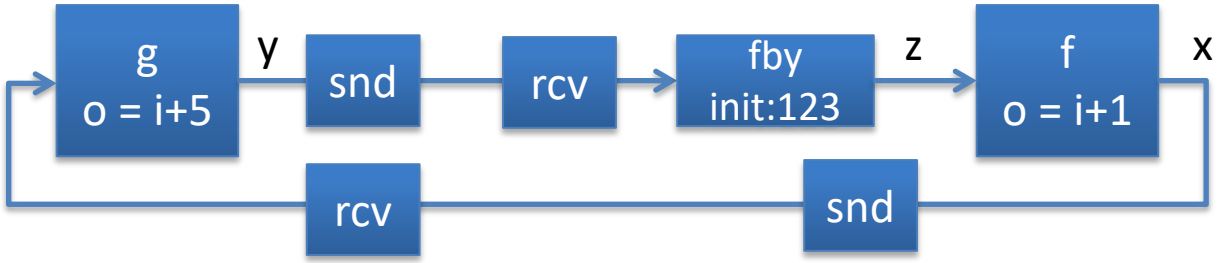
- Specification fonctionnelle

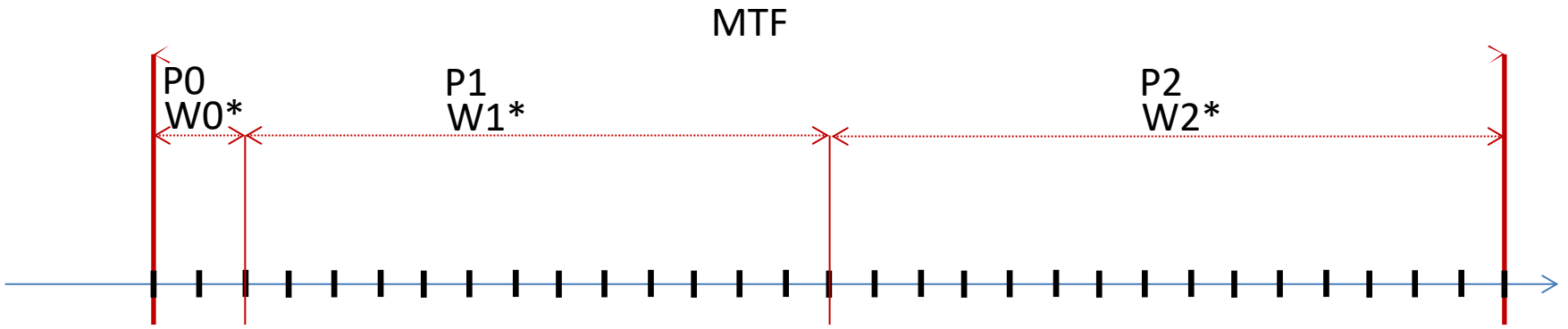




Objectif d'ordonnancement

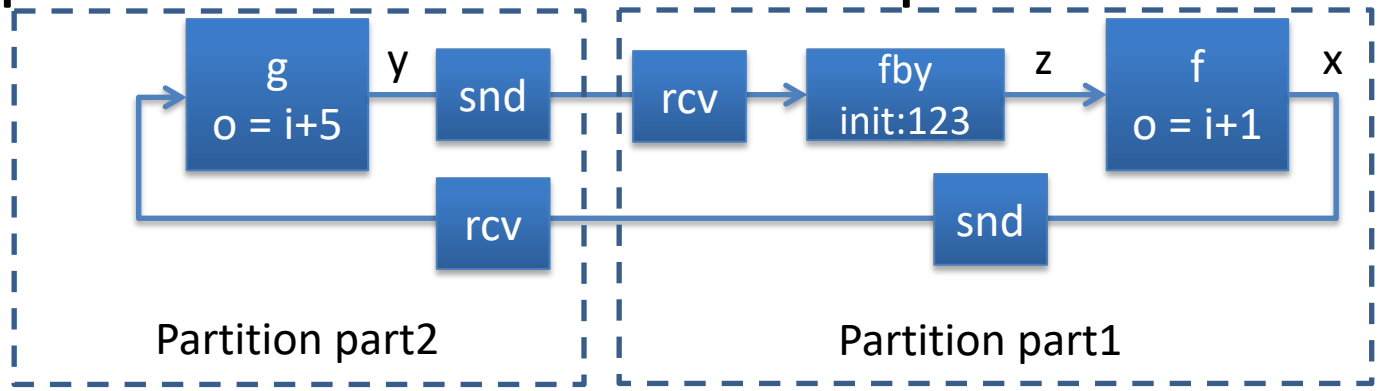
- Specification fonctionnelle + communications

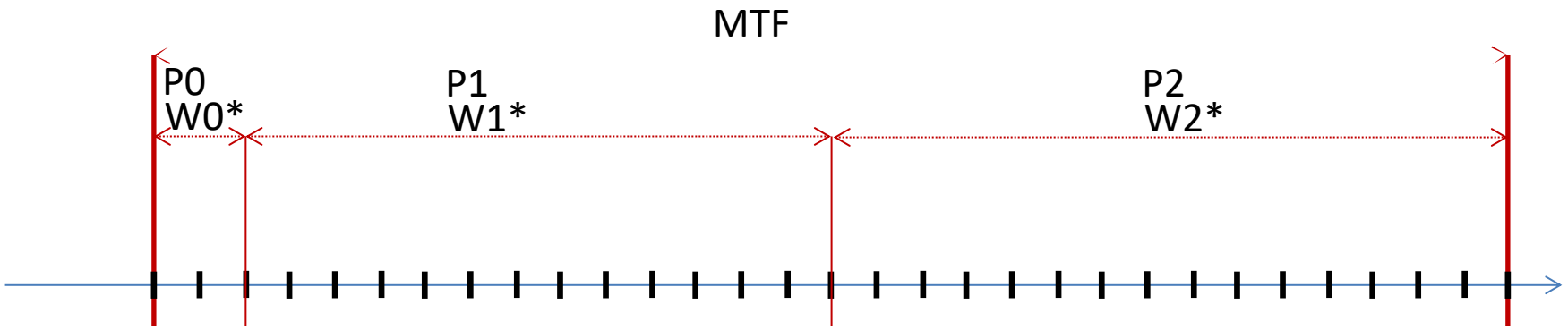




Objectif d'ordonnancement

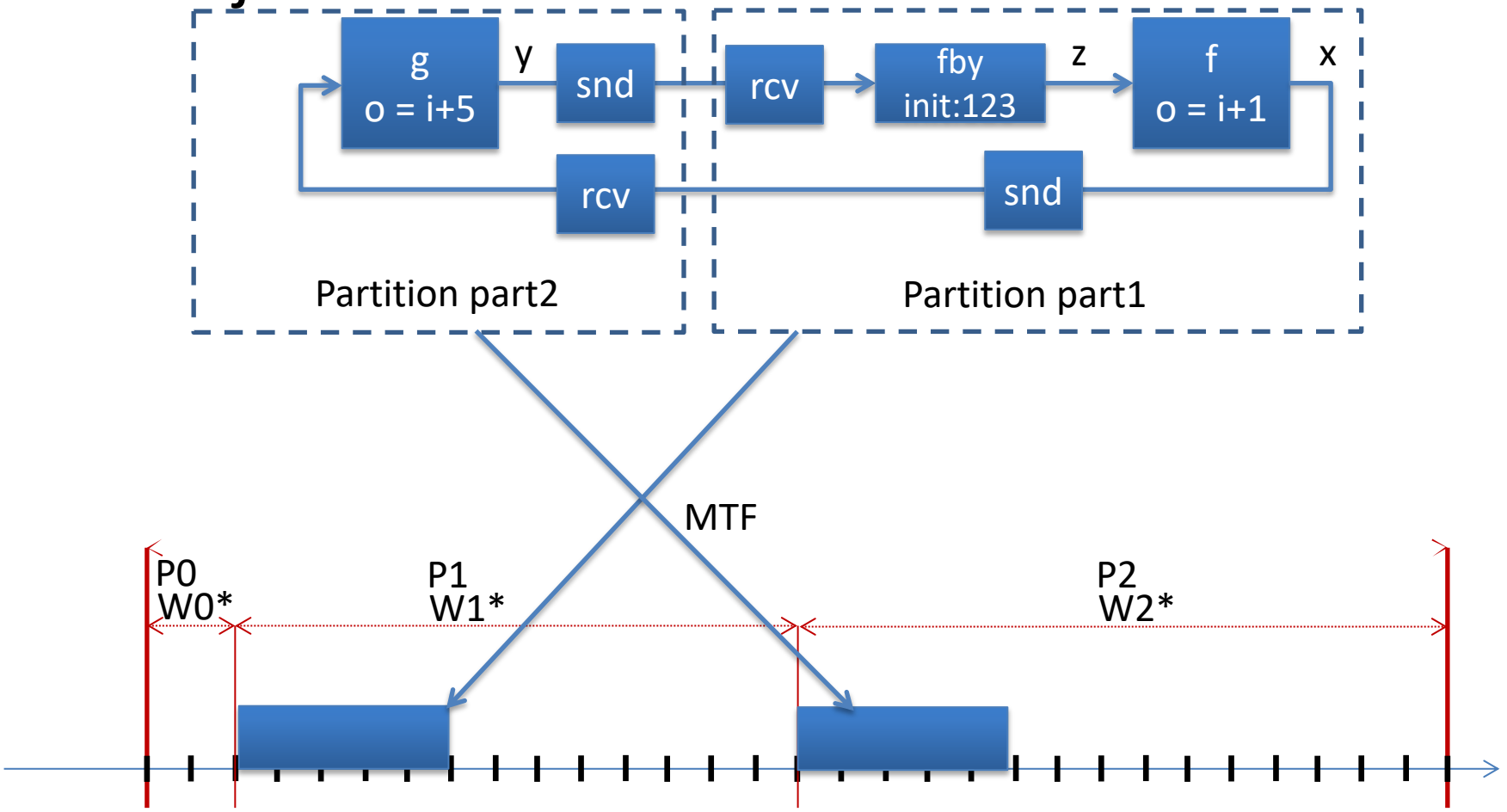
- Specification fonctionnelle partitionnée





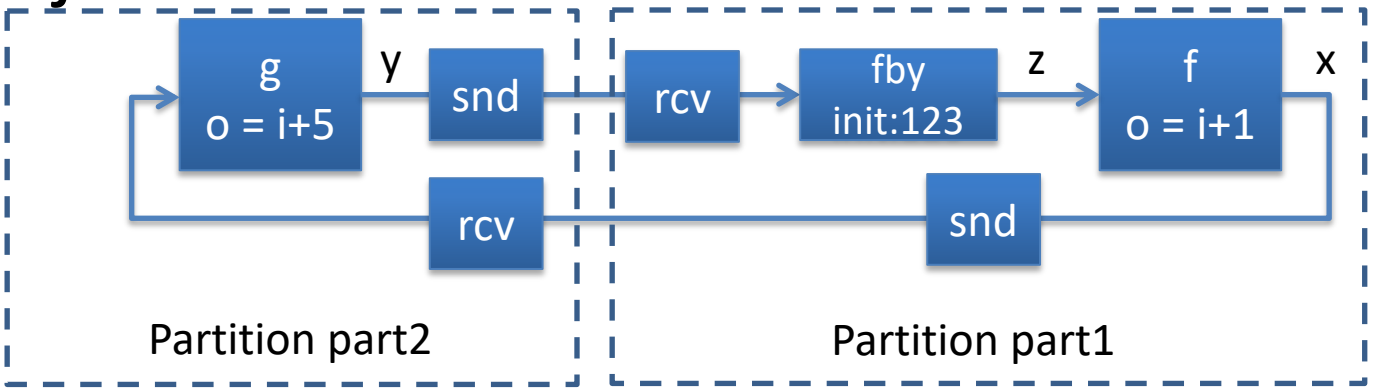
Objectif d'ordonnancement

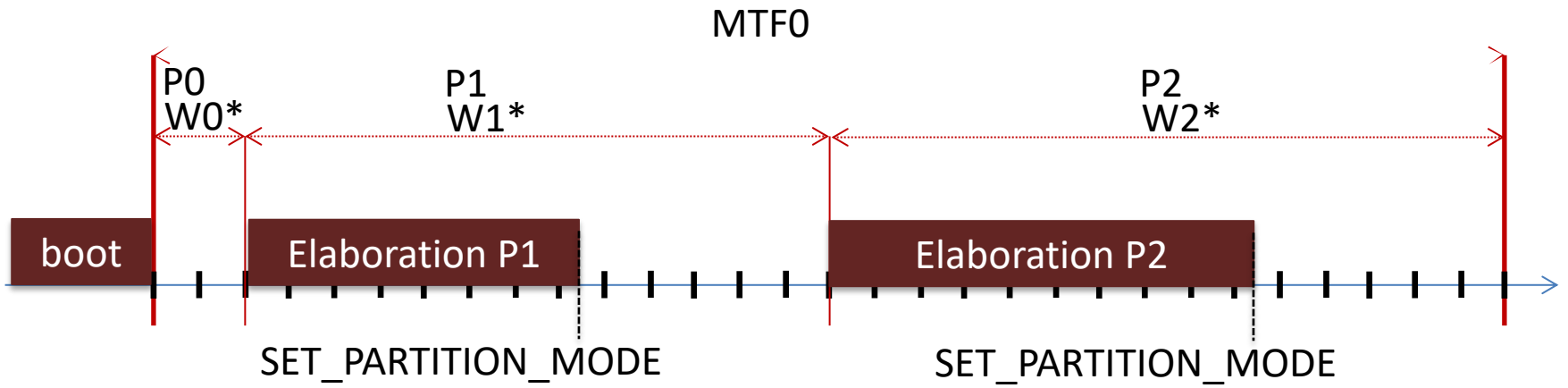
- Objectif de construction de tâches



Initialisation du système

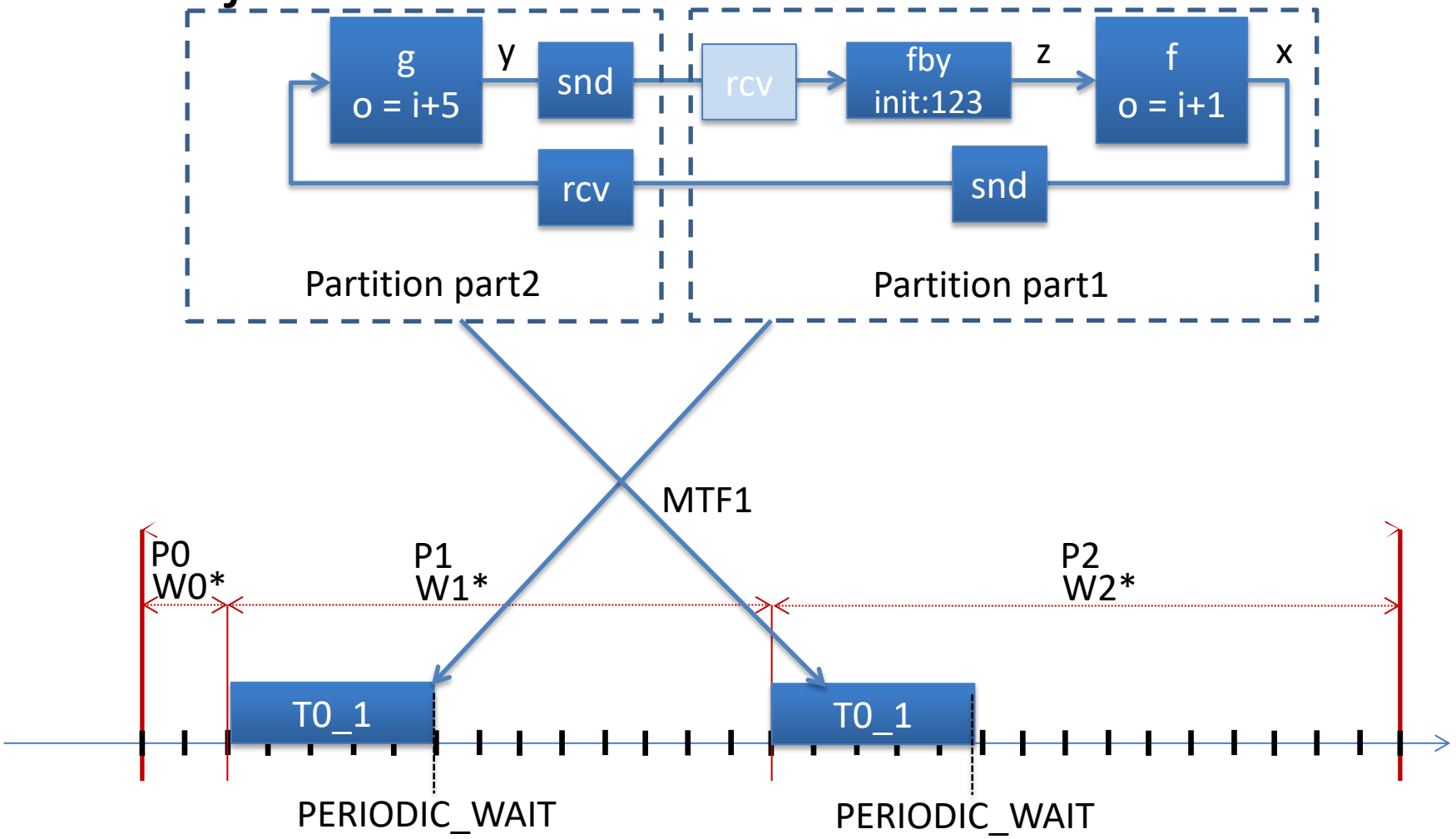
- Objectif de construction de tâches





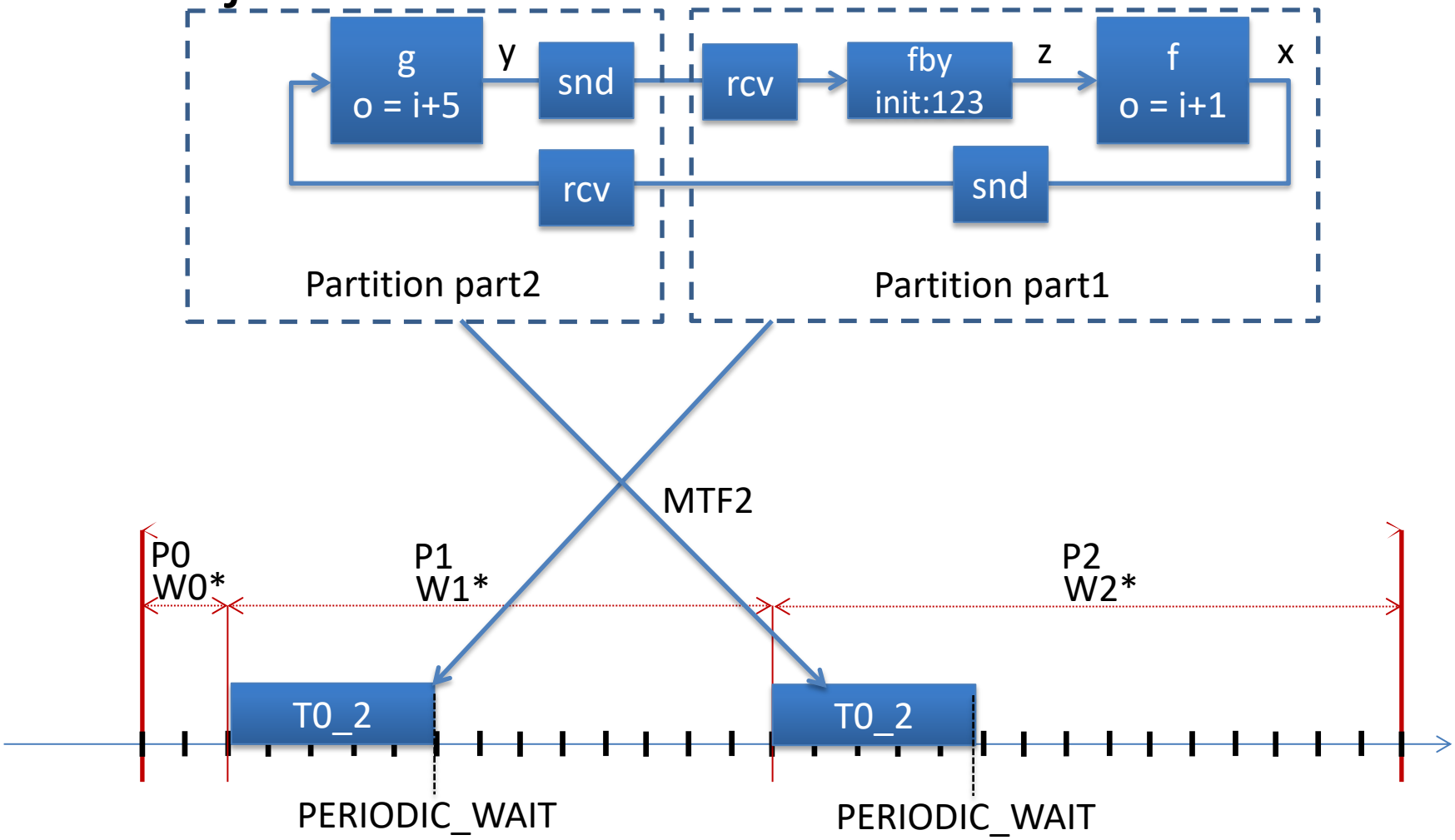
Initialisation du système

- Objectif de construction de tâches



Initialisation du système

- Objectif de construction de tâches



Le démarrage d'une partition

```
// Configuration d'un processus
pat.PERIOD = 0xf0000 ; /* aperiodic process */
pat.TIME_CAPACITY = 0xf0000 ;
pat.ENTRY_POINT = (void*)task0;
pat.STACK_SIZE = 0x1000 ;
pat.BASE_PRIORITY = 0x1 ; /* all tasks have the same */
pat.DEADLINE = HARD ;
strcpy(pat.NAME,"P1Task0") ;
// Création du processus
CREATE_PROCESS(&pat,&pid,&rc) ;
console_perror(rc,"part1","main_process CREATE_PROCESS");
// Démarrage
START(pid,&rc) ;
console_perror(rc,"part1","main_process START");
// Passage en mode préemptif
SET_PARTITION_MODE(NORMAL,&rc) ;
console_perror(rc,"part1","main_process SET_PARTITION_MODE");
```

Organisation d'une tâche simple

```
void task0(){
  // Déclarations et initialisations
  ...
  for(;;){
    // Calcul périodique – attention, certains calculs ou
    // communications sont conditionnels
    ...
    // Fin d'une période, attente de la prochaine
    PERIODIC_WAIT(&rc) ;
    console_perror(rc,"part1","task0 PERIODIC_WAIT");
  }
}
```

Envoi d'un entier

```
int val ;
...
SEND_QUEUING_MESSAGE(0,                 // port 0 of partition 1
                     (APEX_BYTE*)&val, // send buffer
                     sizeof(int),      // size of data to send
                     0,                // non-blocking
                     &rc) ;
console_perror(rc,"part1","task0 SEND");
```

- RPi653 – pas besoin d'initialiser les ports I/O
  - Ils ont les identifiants du fichier config.pok

Réception d'un entier

```
int val ;
...
char recv_buf[sizeof(int)]; // Receive buffer
int recv_size ;             // Size of received data
RECEIVE_QUEUING_MESSAGE(1,  // port 1 of partition 1
                        0,  // nonblocking
                        (APEX_BYTE*)recv_buf,
                        &recv_size,
                        &rc);
console_perror(rc,"part1","task0 RECEIVE");
memcpy((char*)&val,recv_buf,sizeof(int));
// maintenant "val" contient la valeur reçue
```

- RPi653 – pas besoin d'initialiser les ports I/O
  - Ils ont les identifiants du fichier config.pok

Préparation du TP

- Objectif 1 : Implémenter manuellement le système décrit dans les transparents 4-21
  - Écrire la spécification fonctionnelle Heptagon et la faire tourner (avec un sleep(1) dans la boucle "for" de main.c)
    - Pour chaque appel de fonction (f/g) imprimer une ligne contenant le nom de la fonction, la valeur d'entrée et celle de sortie
      - Fonctions externes C + fichier .epi
  - Créer un système à deux partitions avec une tâche périodique par partition
  - Ajouter les deux canaux de communication
  - Écrire (directement en C) le code de calcul périodique des deux tâches de manière à implanter le même comportement que le code Heptagon
    - Mêmes messages attendus en sortie