Trampoline
Un système d'exploitation temps réel pour l'informatique embarquée

Jean-Luc Béchennec, Mikaël Briday, Sébastien Faucou

LS2N – UMR CNRS 6004
CNRS, École Centrale de Nantes, École des Mines de Nantes, Université de Nantes

École d'été Temps Réel – ETR 2021
Poitiers, 22 septembre 2021

Section 1

Contexte, définition

Système embarqué

Système électronique et informatique intégré à un système mécatronique en vue de piloter ses organes

Principales caractéristiques

Ressources matérielles limitées

Matériel adapté aux contraintes opérationnelles (thermiques, vibratoires, etc) et économiques $\longrightarrow$ micro-contrôleur

Temps réel

- Contraintes *hard*, *firm*, *soft*
- Dynamique parfois très rapide : les temps de réponse exigés peuvent s'exprimer en centaines de micro-secondes

Contexte industriel

- Normes, standards, certification

Architecture matérielle type : caractéristiques

- Microcontrôleur
- Fréquence $<$ 500 MHz et généralement $<$ 200 MHz
- Mono ou Multicœur
- 8 / 16 / 32 bits, préférablement RISC
- Cache : aucun ou L1 uniquement, souvent I-Cache uniquement
- Protection mémoire : rien, MPU, plus rarement MMU
- Pas d'assistance matérielle à la virtualisation

Architecture fonctionnelle type

Application = ensemble de fonctions temps réel communicantes

Architecture fonctionnelle type

Application = ensemble de fonctions temps réel communicantes

Fonctions

- Partie commande : régulateurs, filtres, machine à états
- Partie système : pilotes, pile de communication, monitoring, ...

Architecture fonctionnelle type

Application = ensemble de fonctions temps réel communicantes

Temps réel

- Activations périodiques ou sporadiques
- Échéances contraintes ou sur requêtes
- Criticités multiples

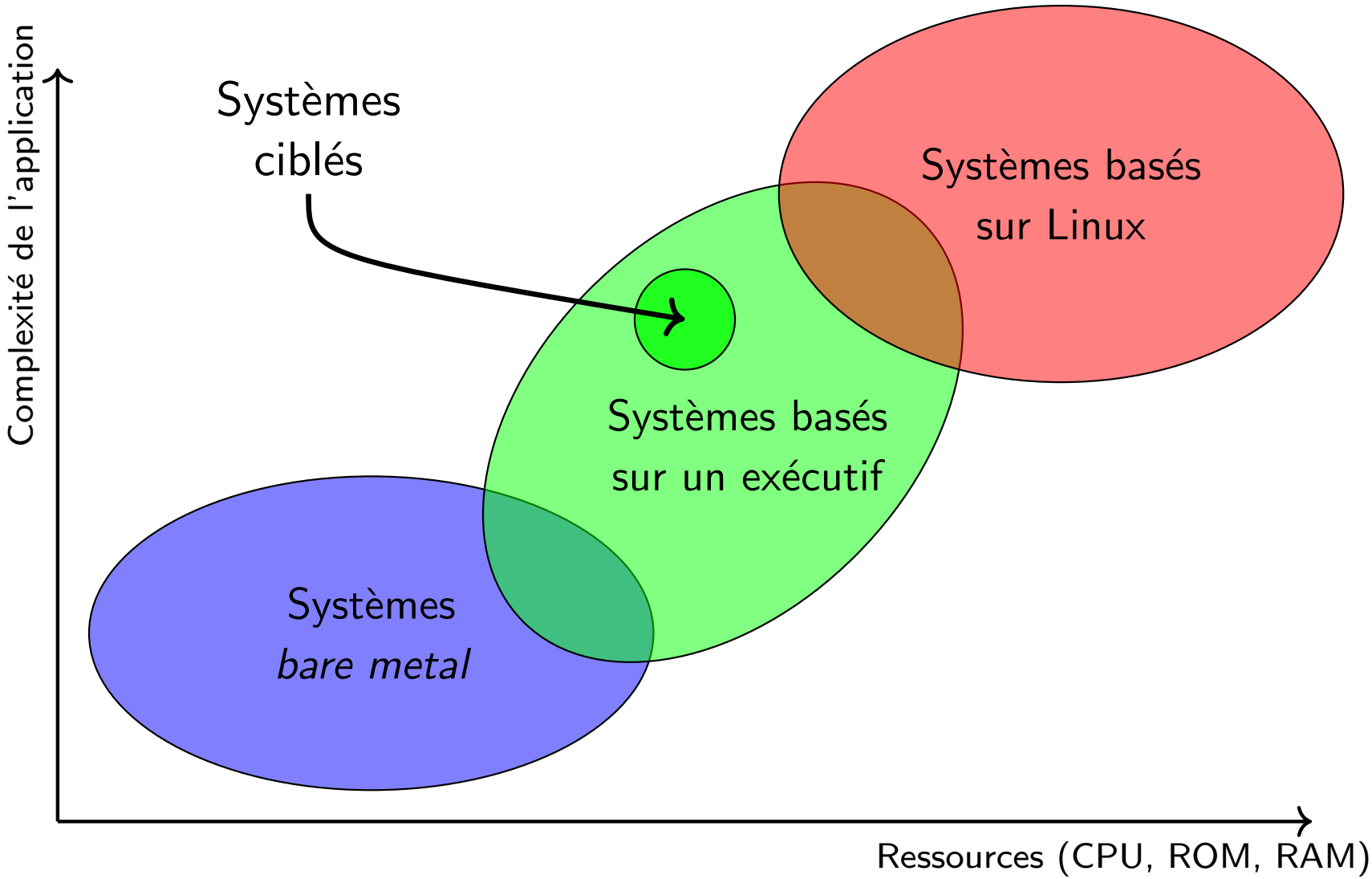
Architecture fonctionnelle type

Application = ensemble de fonctions temps réel communicantes

Communication (et synchronisation)

- Signaux booléens ou numériques, scalaires ou vectoriels
- Utilisation de queues de messages et/ou de blackboard
- Ressources partagées (périphériques)

Positionnement



Système d'exploitation temps réel

Système d'exploitation temps réel (SETR)

Un système d'exploitation (SE) est qualifié de temps réel s'il permet de construire un système aux temps de réponses **prédictibles** et **compatibles avec les dynamiques de l'environnement.**

Chemin critique

- Prise en compte des interruptions
- Ordonnancement
- Changement de contexte

Impact sur la conception

Déterminisme

- Restriction du nombre de services en mode noyau

Prédictibilité

- Éviction des algorithmes / services dont le pire cas est pathologique
- Minimiser le pire-cas, éventuellement au détriment d'autres objectifs : équité, flexibilité, sécurité, etc.

Les services de base

Temps réel

- Ordonnancement
- IPC : synchronisation, signalisation, communication
- Prise en compte des interruptions
- (Mesure du temps : horloges, réveils, chien de garde)

Tolérance aux fautes

- Isolation mémoire (quand le matériel le permet)
- Isolation temporelle

Et le reste ?

Reléguée hors du noyau

- Piles de communication
- Pilotes de périphériques

Parfois/souvent/toujours inutiles

- Système de gestion de fichiers
- Gestion des utilisateurs

Difficilement temps réel

- Mémoire virtuelle (*swapping*)

Section 2

Trampoline : un exécutif temps réel

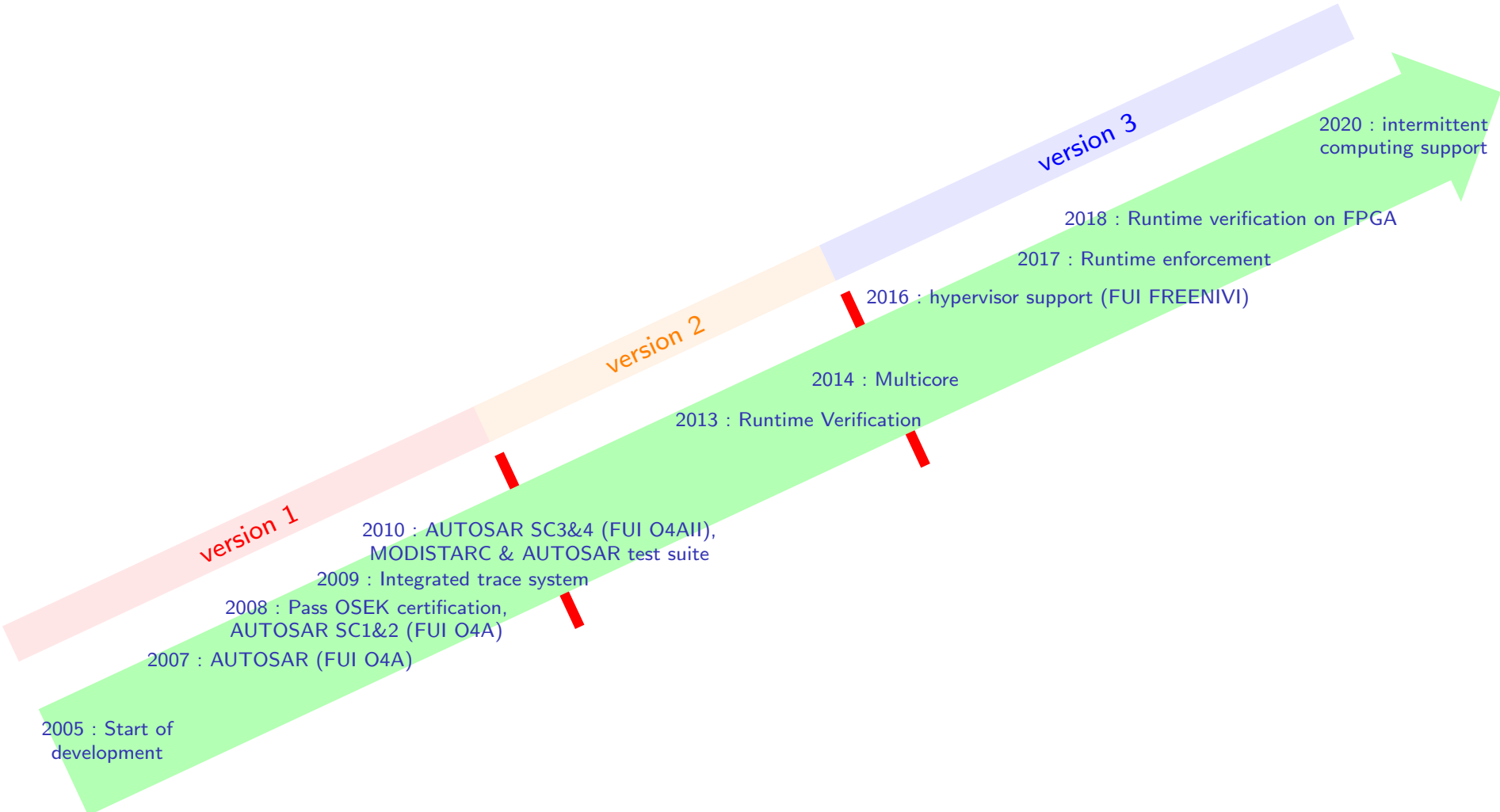
Trampoline : un exécutif temps réel

- Présentation
- Processus de construction d'une application
- Architecture interne

Trampoline

- Exécutif temps réel, développé par l'équipe Systèmes Temps Réel du LS2N (ex-IRCCyN)
- Modulaire et portable
- Licence libre (GPL) + version industrialisée
- Conforme OSEK/VDX OS et AUTOSAR OS
- Nombreuses cibles dont Arduino, Posix, PowerPC, ARM Cortex-M, ...
- Support des architectures multicœurs
- ROM : 5 à 10 ko, RAM : $\sim$ 30 o + $\sim$ 15 o / tâche ;

Histoire


version 1
version 2
version 3
2005 : Start of development
2007 : AUTOSAR (FUI O4A)
2008 : Pass OSEK certification, AUTOSAR SC1&2 (FUI O4A)
2009 : Integrated trace system
2010 : AUTOSAR SC3&4 (FUI O4AII), MODISTARC & AUTOSAR test suite
2013 : Runtime Verification
2014 : Multicore
2016 : hypervisor support (FUI FREENIVI)
2017 : Runtime enforcement
2018 : Runtime verification on FPGA
2020 : intermittent computing support

Utiliser Trampoline : pourquoi ? comment ?

Pourquoi ?

Disposer d'un composant libre pour construire des bancs d'essais

Support d'enseignement

Expérimenter des nouveaux algorithmes / services

Comment ?

Dépôt : `https://github.com/TrampolineRTOS/trampoline`

Système statique

Un système d'exploitation est statique s'il n'offre pas la possibilité de créer des objets (tâche, ressource, boîte aux lettres, etc.) en ligne. Il doit donc être entièrement configuré à la compilation.

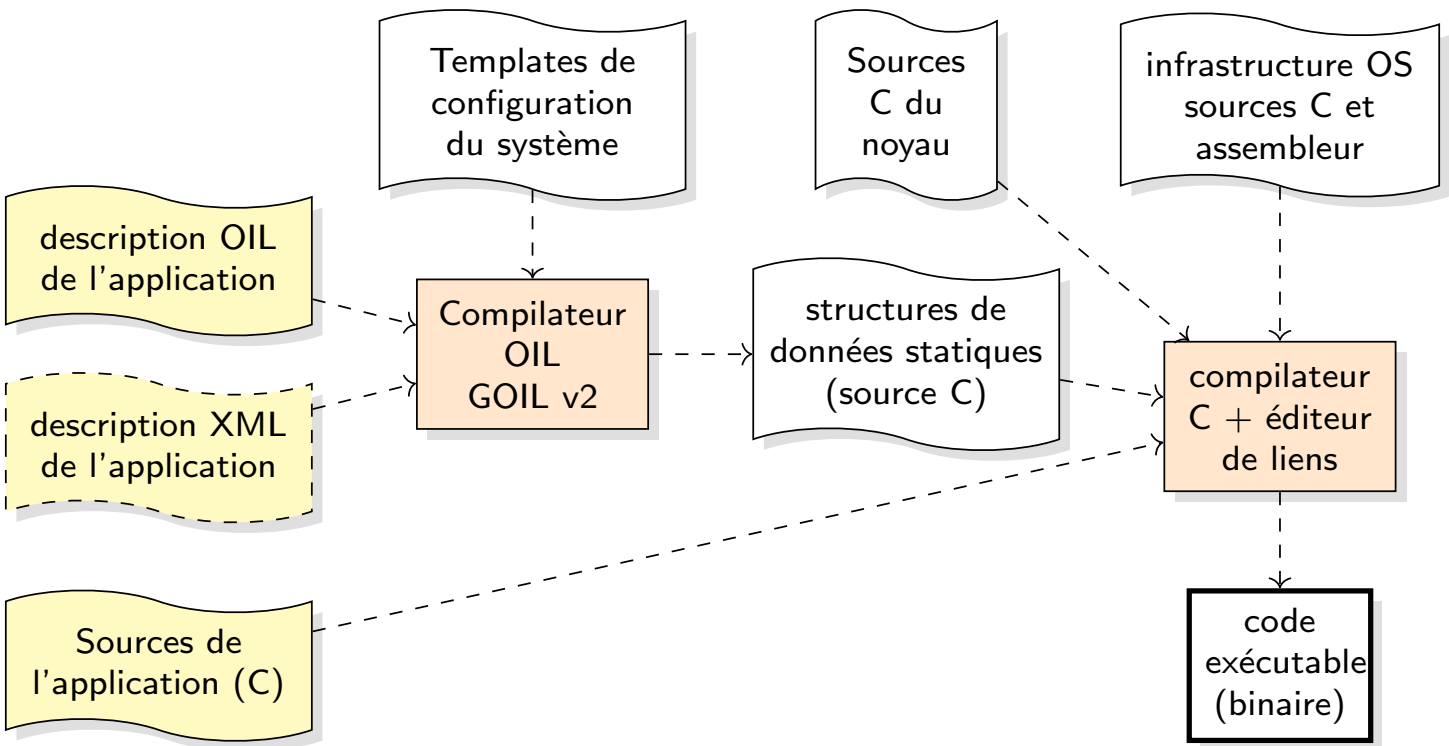
Le noyau d'un système statique peut être « taillé sur mesure » pour l'application

- Services à inclure
- Structure de données
- Et plus encore : chemins d'exécution, variables, etc.

Trampoline est un système statique

Trampoline utilise un langage dédié

- dérivé de la norme OIL
- extensible (basé sur un moteur de template)



Exemple de description

```
ALARM one_second {
  COUNTER = SystemCounter;
  ACTION = ACTIVATETASK { TASK = t0; };
  AUTOSTART = TRUE {
      APPMODE = std;
      ALARMTIME = 100;
      CYCLETIME = 100;
  };
};

TASK t0 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  MESSAGE = s00;
};

MESSAGE s00 {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
      CDATATYPE = "uint8";
  };
  NOTIFICATION = NONE;
};
```

Exemple de template goil

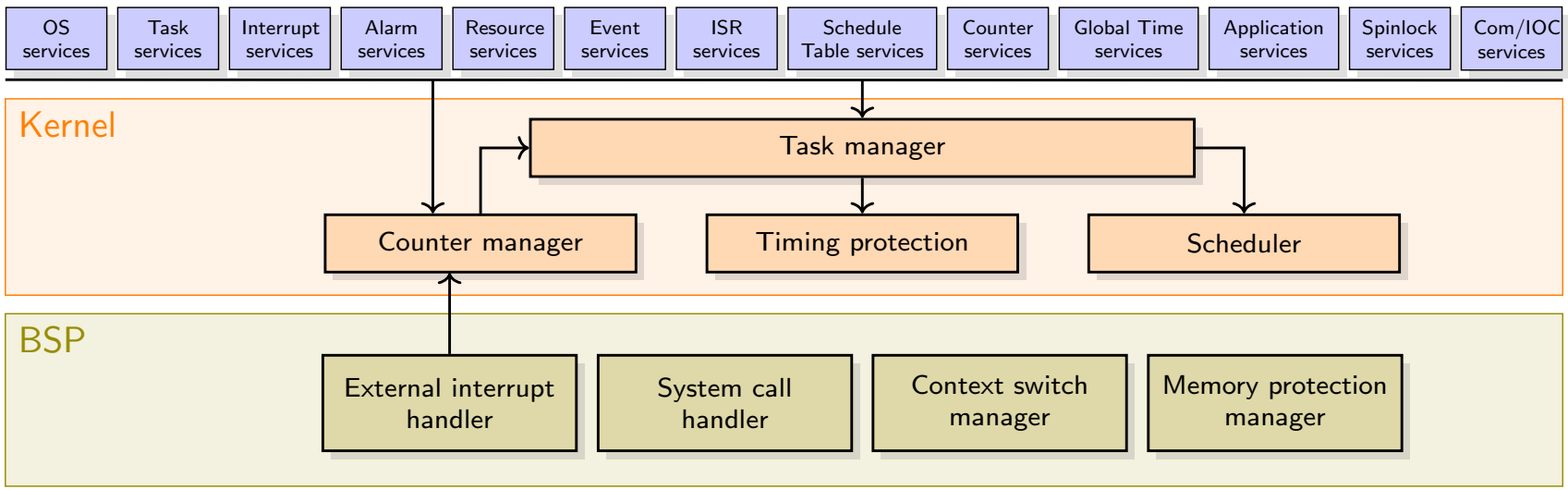
```
/*============================================================
 * Definition and initialization of process tables (tasks and isrs)
 */
CONSTP2CONST(tpl_proc_static, AUTOMATIC, OS_APPL_DATA)
tpl_stat_proc_table[TASK_COUNT+ISR_COUNT+% ! OS::NUMBER_OF_CORES %] = {
%
foreach proc in PROCESSES do
% &% !proc::NAME %_% ![proc::KIND lowercaseString] %_stat_desc,
%
end foreach
if OS::NUMBER_OF_CORES == 1 then
% &IDLE_TASK_task_stat_desc%
else
  loop core from 0 to OS::NUMBER_OF_CORES - 1 do
    %  IDLE_TASK_% ! core %_task_stat_desc%
  between %,
%
  end loop
end if
%

};
```

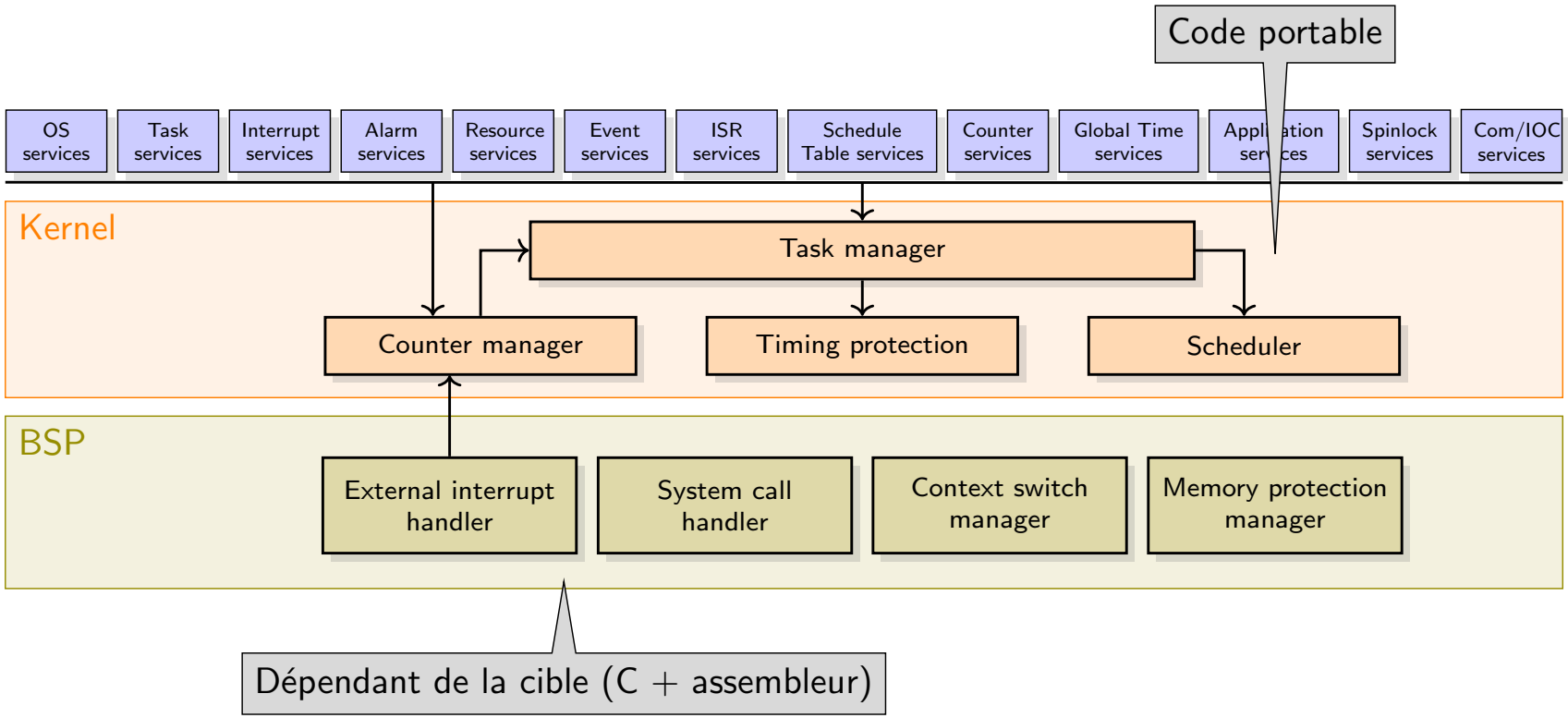
Avantages de `goil` sur une API

- Séparation des préoccupations
- Niveau d'abstraction
- Plus simple à faire évoluer
- Automatisation de la spécialisation du noyau
- Génération automatique : Code, Makefile, *link script*

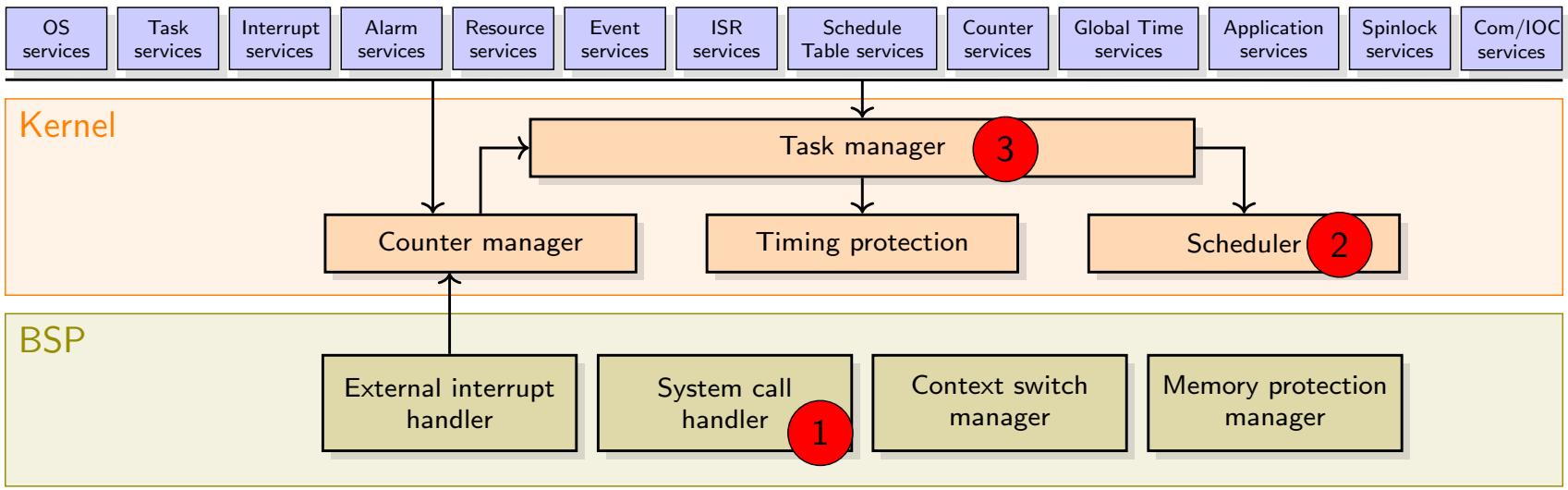
Architecture



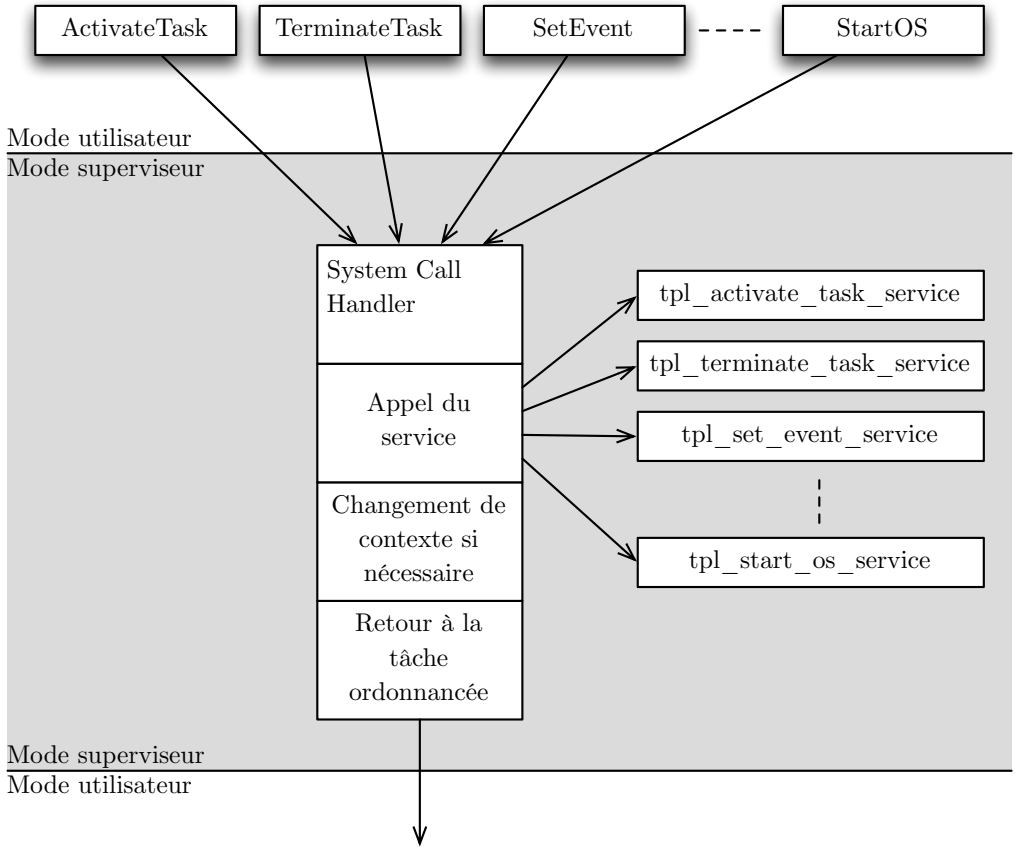
Architecture



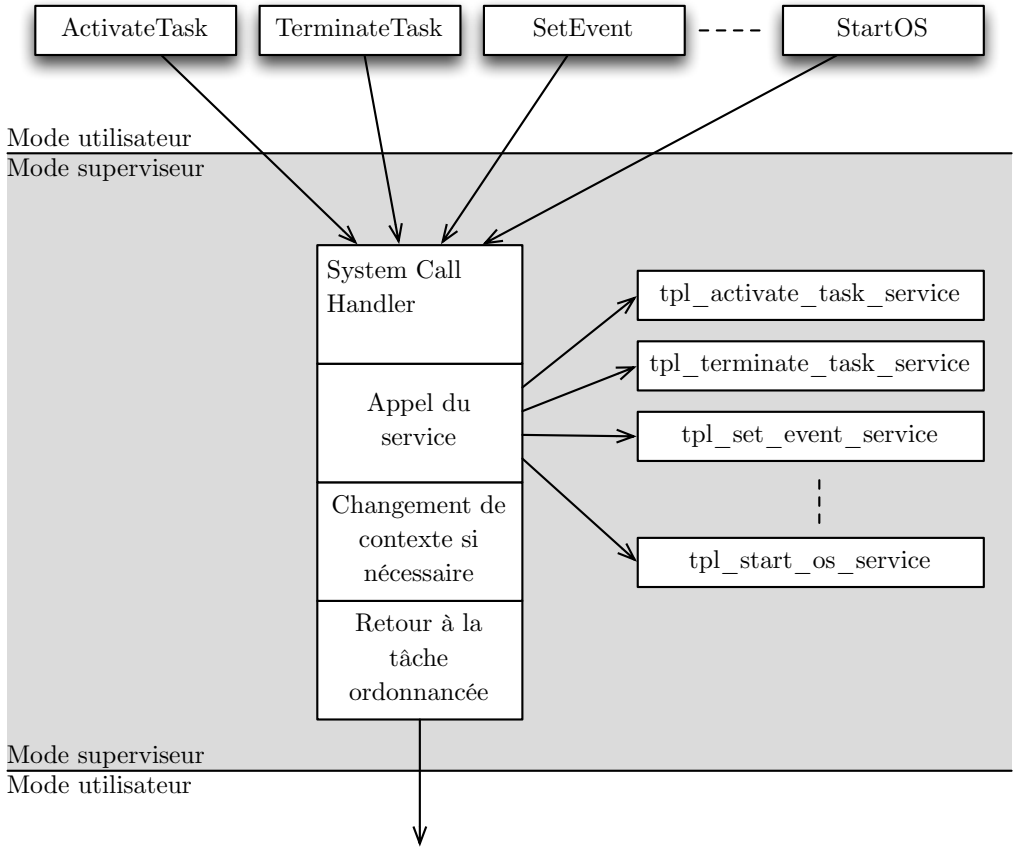
Architecture



Module *System Call Handler*



Module *System Call Handler*



Une fois passé en mode superviseur, le noyau est non préemptible.
Il faut donc minimiser la durée de ce passage pour assurer la réactivité du système

Module *System Call Handler*

Passage en mode superviseur (exemple sur PowerPC)

```
  .global SuspendOSInterrupts
SuspendOSInterrupts:
  li   r0,OSServiceId_SuspendOSInterrupts  /* service id in r0 */
  sc                                       /* system call      */
  blr                                      /* returns          */
```

Selon l'ABI utilisée, les paramètres et le code de retour de l'appel système sont passés dans des registres ou sur la pile

Module *Scheduler*

Ordonnanceur

Assure la manipulation de la liste des tâches prêtes grâce à quatre fonctions :

- `tpl_put_new_proc` : Ajout d'une tâche dans la liste des tâches prêtes ;
- `tpl_put_preempted_proc` : Ajout d'une tâche préemptée ;
- `tpl_front_proc` : obtenir la tâche la plus prioritaire ;
- `tpl_remove_front_proc` : enlever la tâche la plus prioritaire.

Module *Scheduler*

Politique d'ordonnancement

FP/FIFO

En pratique, toute politique *fixed job priority* qui assure un ordre *FIFO* entre les *jobs* successifs d'une même tâche peut être implémentée

Modifier la politique d'ordonnancement

1. Ajouter les informations nécessaires aux descripteurs de tâches à l'aide des *templates* `goil`
2. Adapter le calcul de la priorité dans `tpl_put_new_proc`
3. Adapter le protocole de synchronisation pour l'accès aux ressources partagées (IPCP par défaut)

Module *Task manager*

Task Manager

Il fournit les fonctions permettant de faire évoluer l'état des « tâches ».

Utilise les fonctions du module *Scheduler* pour manipuler les *jobs* engendrés.

Tâche ?

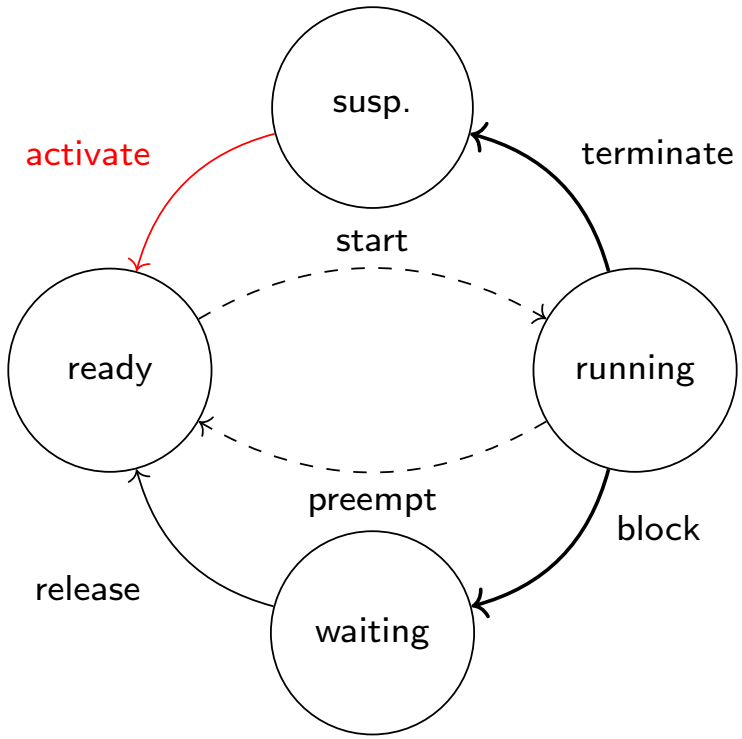
L'API propose 3 types de tâches

- tâche basique : pas d'appel bloquant
- tâche étendue
- ISR2 : similaire aux tâches basiques mais activée sur une IRQ

En interne, un seul type est utilisé : `tpl_proc`

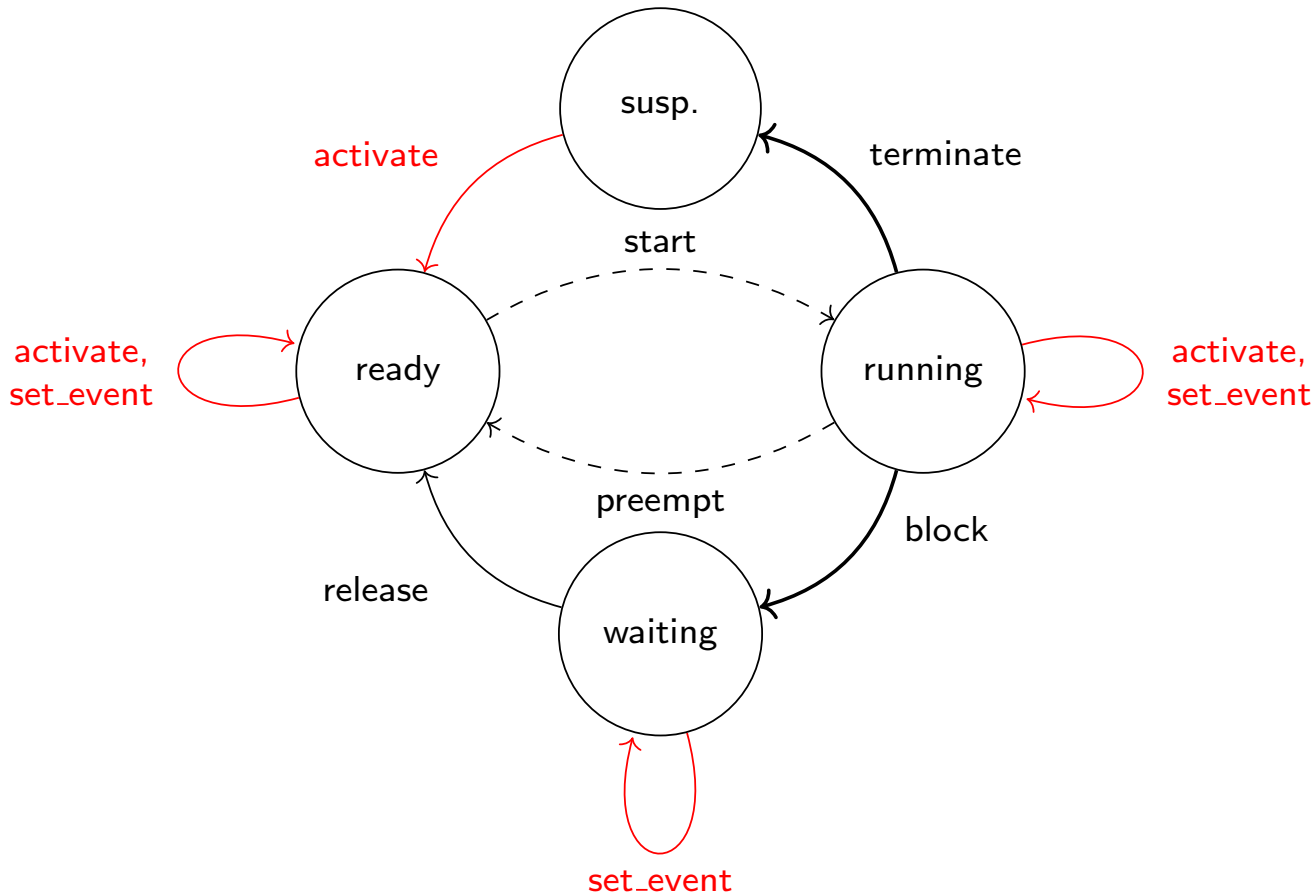
Module *Task Manager*

Le module *Task Manager* fait évoluer l'état des objets `tpl_proc`



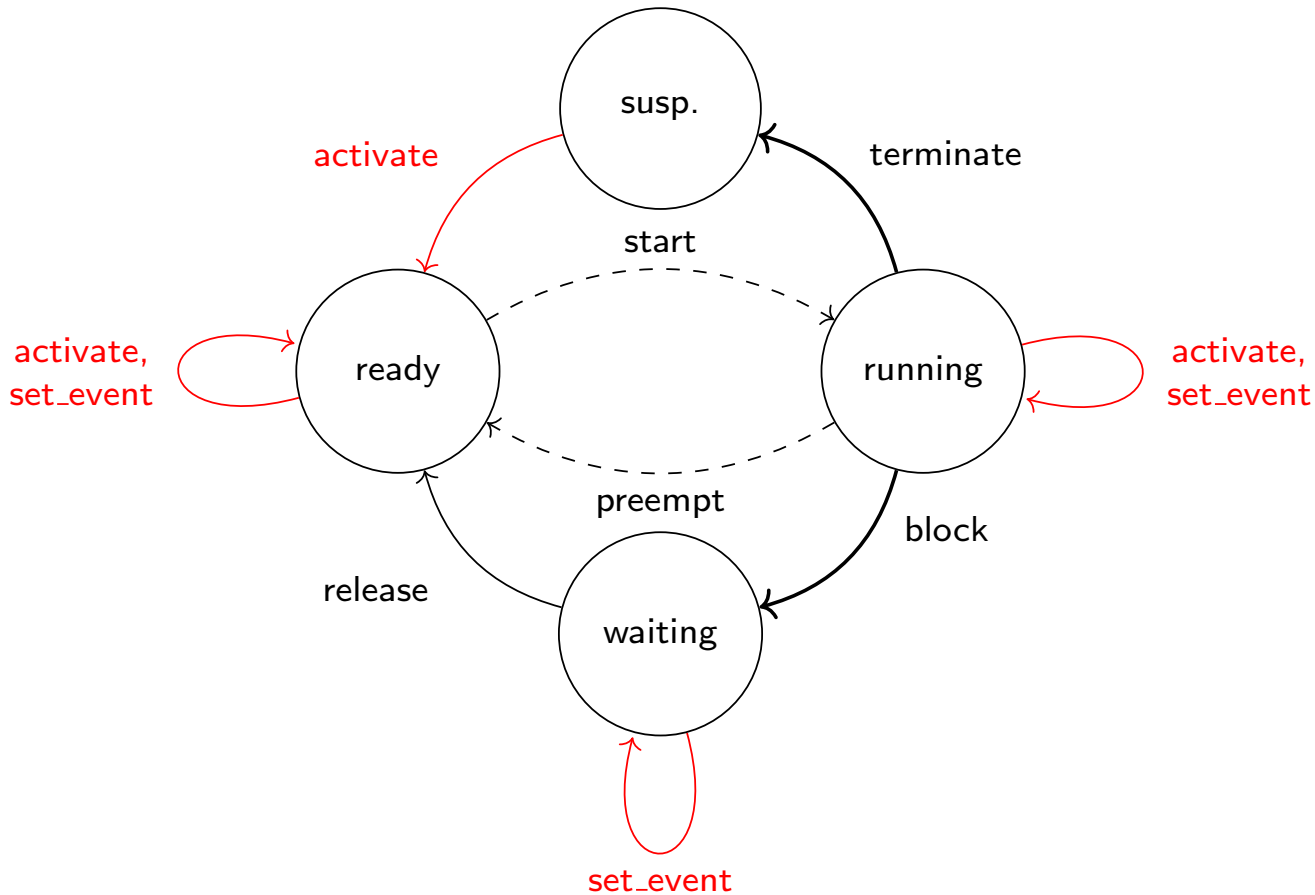
Module *Task Manager*

Le module *Task Manager* fait évoluer l'état des objets `tpl_proc`



Module *Task Manager*

Le module *Task Manager* fait évoluer l'état des objets `tpl_proc`



- Enregistrement des activations jusqu'à une borne fixée
- Enregistrement des événements dans un vecteur de bit

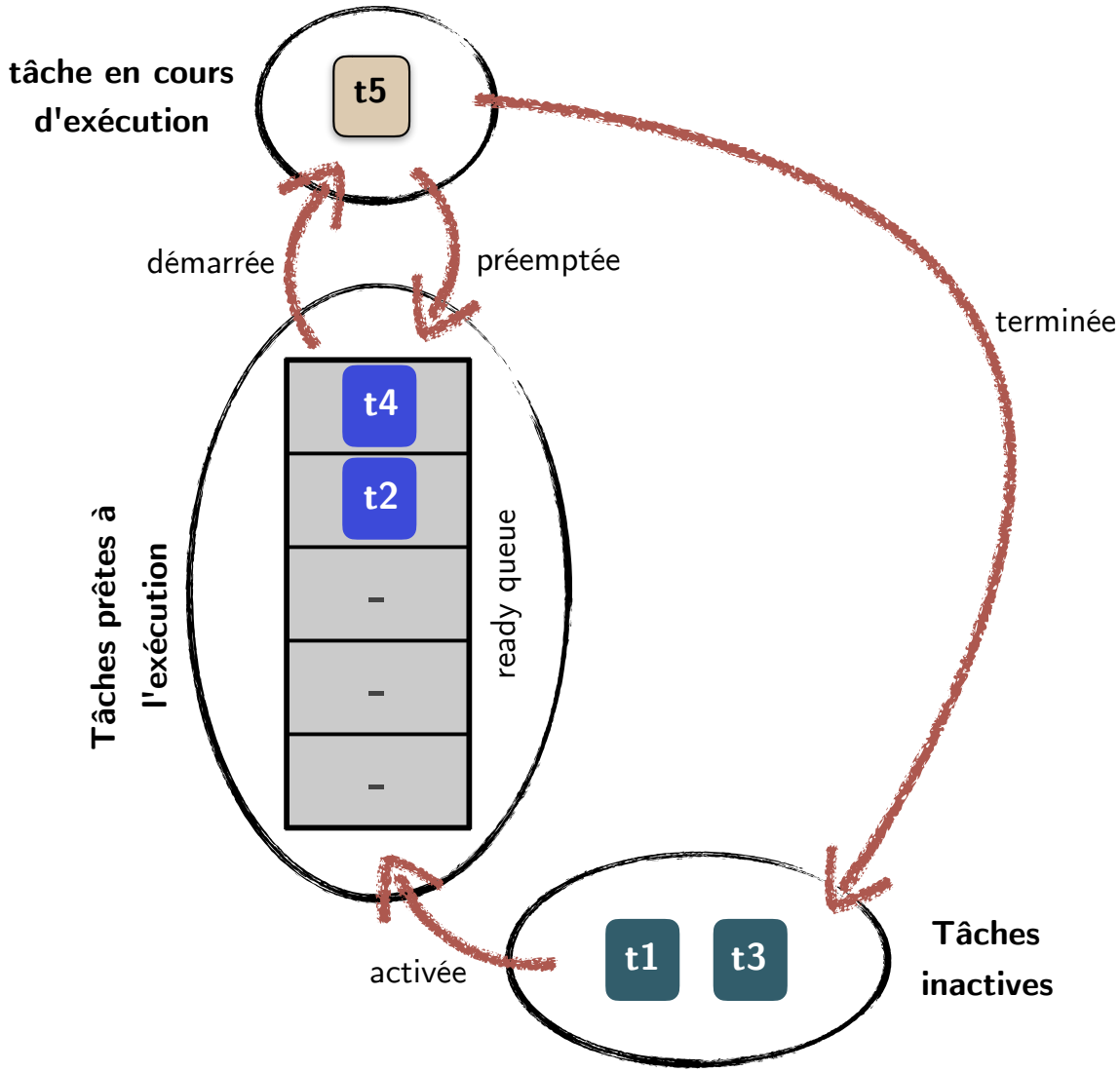
Module *Task Manager*

L'API du module se déduit du modèle

- `tpl_activate_task`
- `tpl_preempt`
- `tpl_start`
- `tpl_terminate`
- `tpl_block`
- `tpl_release`
- `tpl_set_event`

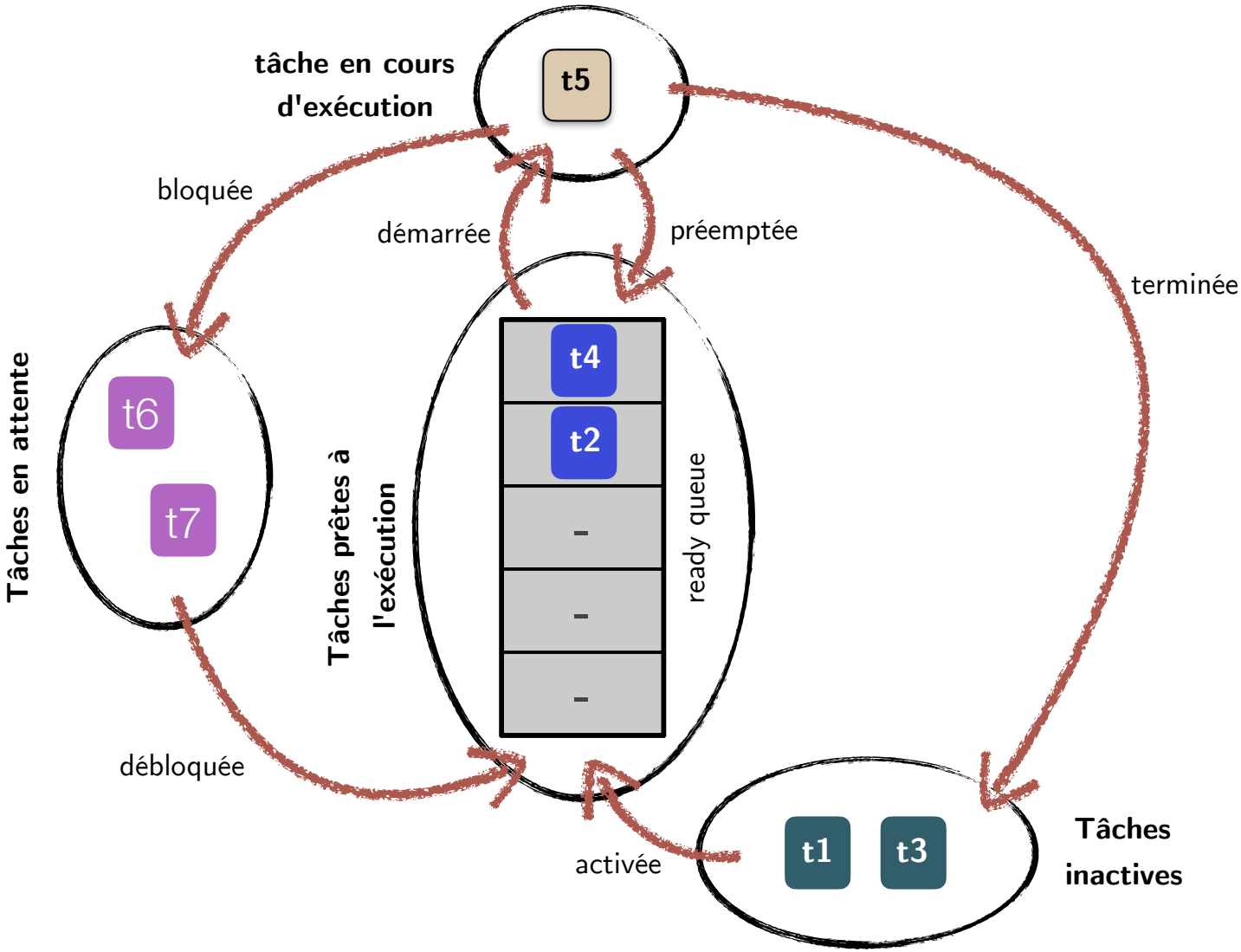
Module *Task Manager*

L'ordonnanceur gère une liste des tâches prêtes :



Module *Task Manager*

Ajout de la capacité à attendre un événement :



Section 3

Focus sur le changement de contexte

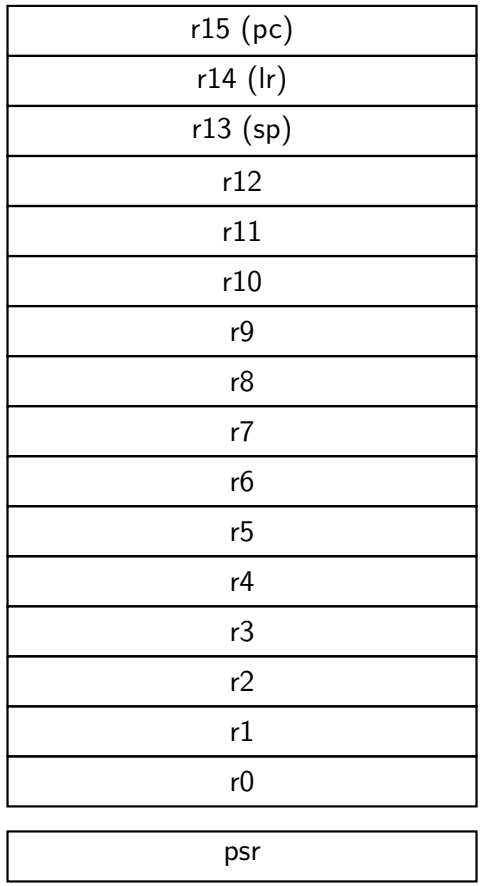
Qu'est-ce qu'un contexte ?

C'est l'ensemble des registres du processeur qui sont propres au programme en cours d'exécution. Par exemple sur ARM.

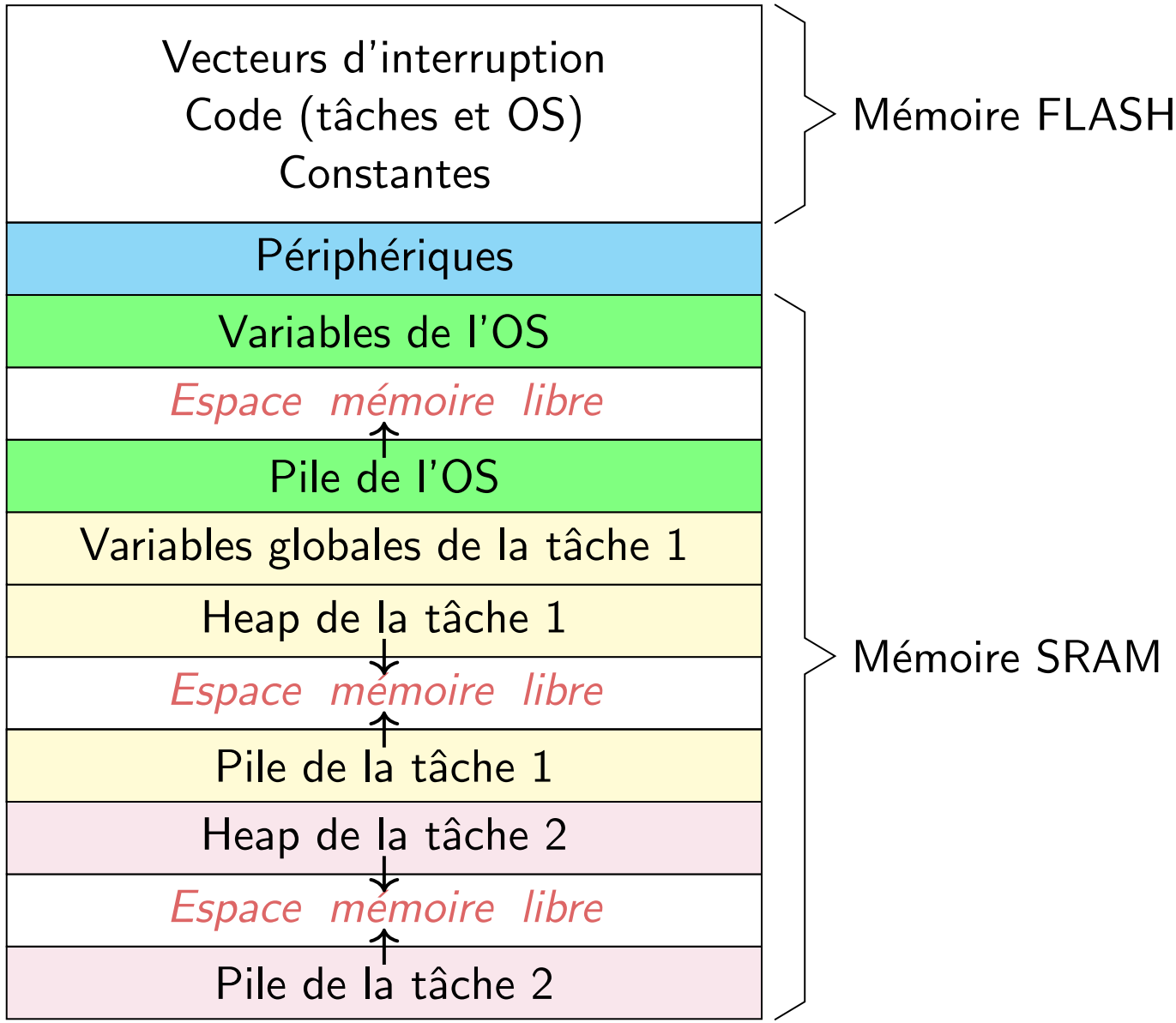
Le CPU utilise 16 registres généraux de 32 bits : r0 à r15. Parmi eux, 3 registres ont une signification particulière :

- r13 est le pointeur de pile, sp
- r14 est le registre de liaison, lr
- r15 est le compteur de programme, pc

Enfin, le psr stocke quelques drapeaux.



Organisation de l'espace mémoire



Comment ceci est implémenté ?

Quand aucune tâche n'est prête, quelque chose doit s'exécuter

- Une tâche cachée *idle* avec une priorité $<$ à toutes les tâches ;
- Ne se termine jamais, ne bloque jamais ;
- Implémente une boucle infinie ;
- Peut optionnellement place le MCU en basse consommation. État d'où il sera tiré par une interruption.

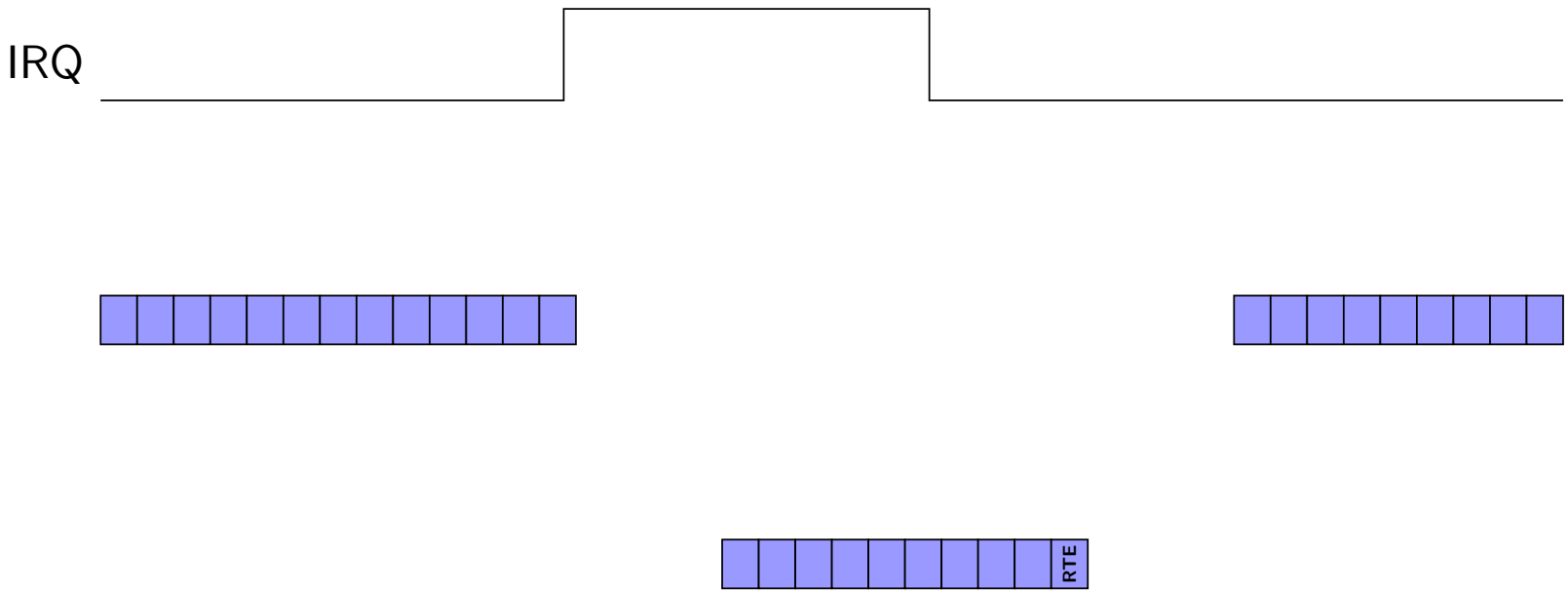
Les tâches périodiques nécessitent une base de temps

- Un timer délivre une *interruption* périodique (systick) ;
- Généralement une période de 1ms est choisie mais peut être ajustée selon les besoins de l'application.

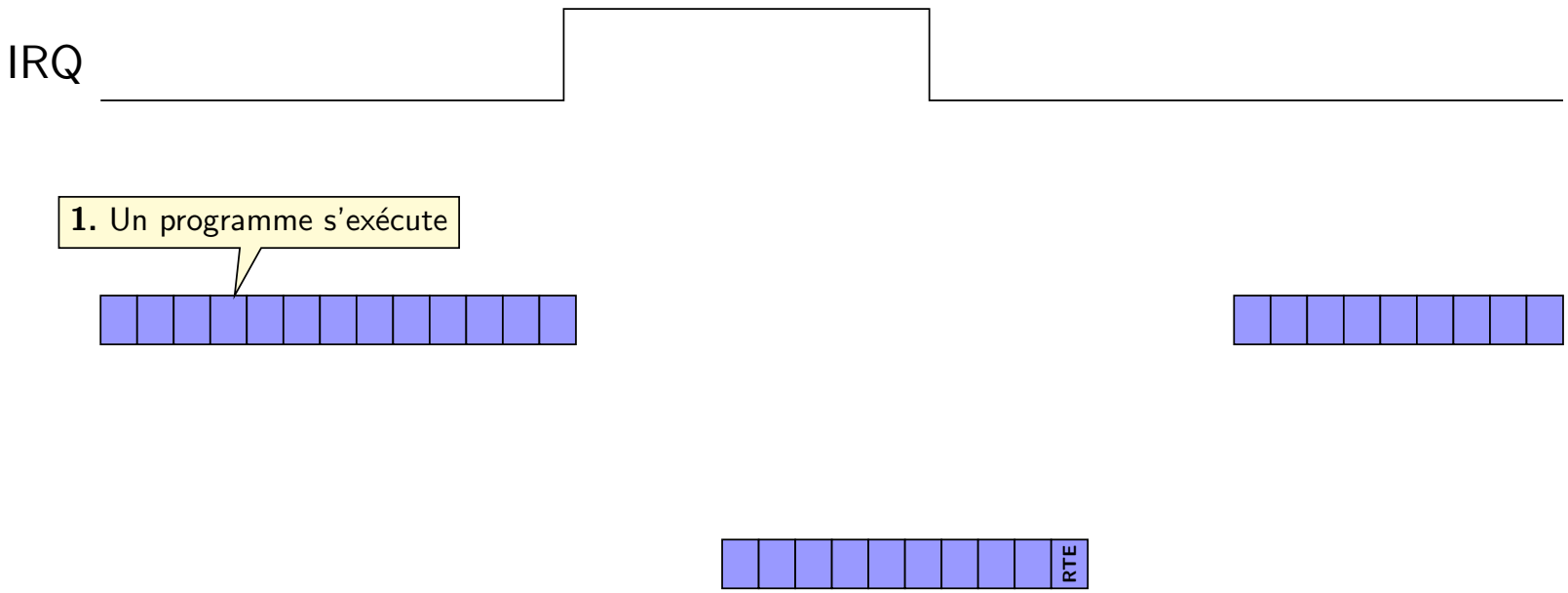
Interruptions (1)

- Du point de vue matériel, une ligne d'interruption est une entrée matérielle du processeur (IRQ, Interrupt ReQuest). L'IRQ est un état ;
- Le processeur peut se trouver dans des états où :
  - l'IRQ est ignorée temporairement (aka masquée) ou
  - l'IRQ est prise en compte (non masquée).
- Lorsque l'IRQ devient active et si l'interruption n'est pas masquée, le programme en cours d'exécution est arrêté et le processeur exécute un autre programme appelé le gestionnaire d'interruptions.
- Lorsque le gestionnaire d'interruption se termine, le processeur revient au programme précédemment arrêté.

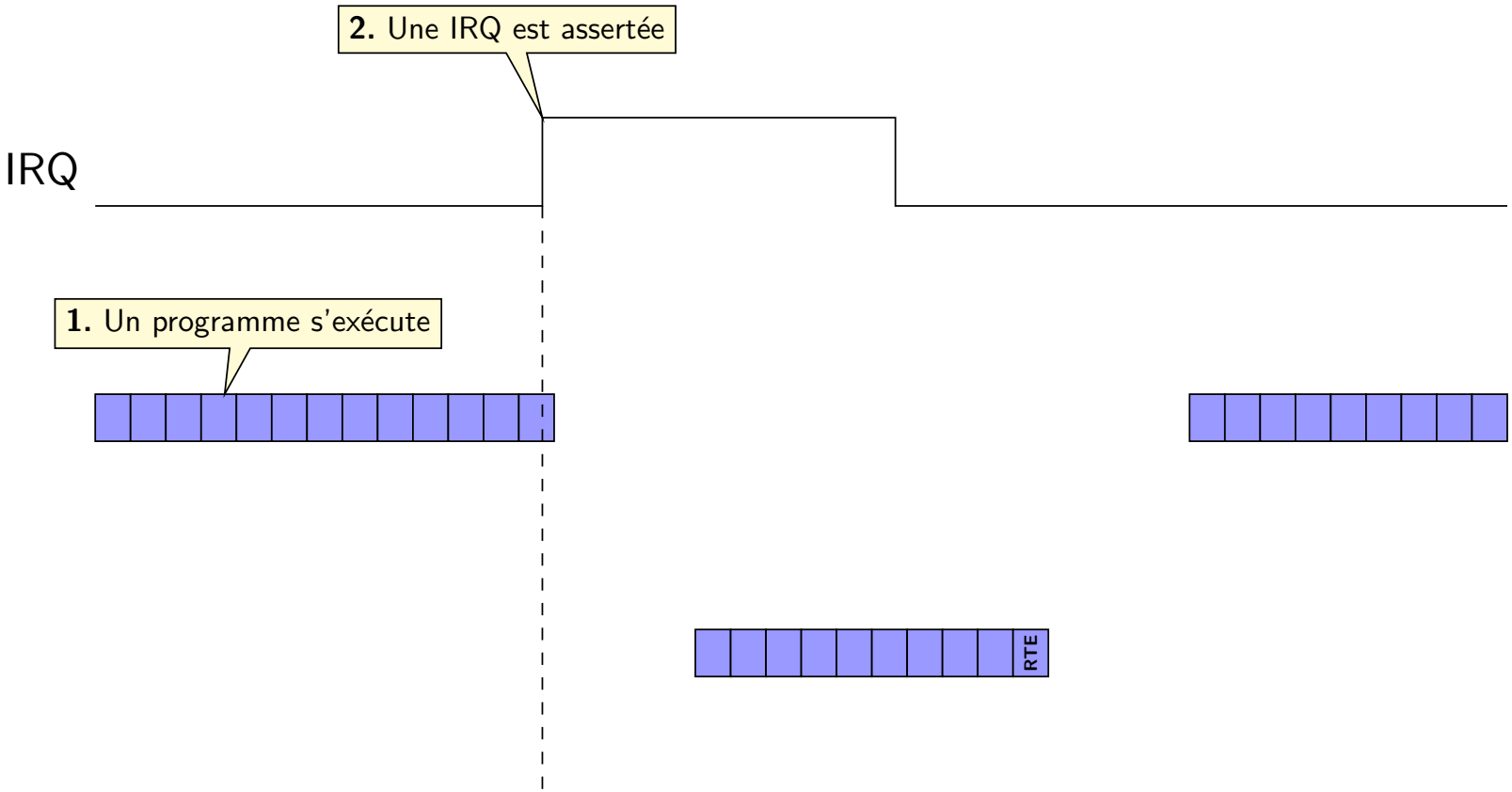
Interruptions (2)



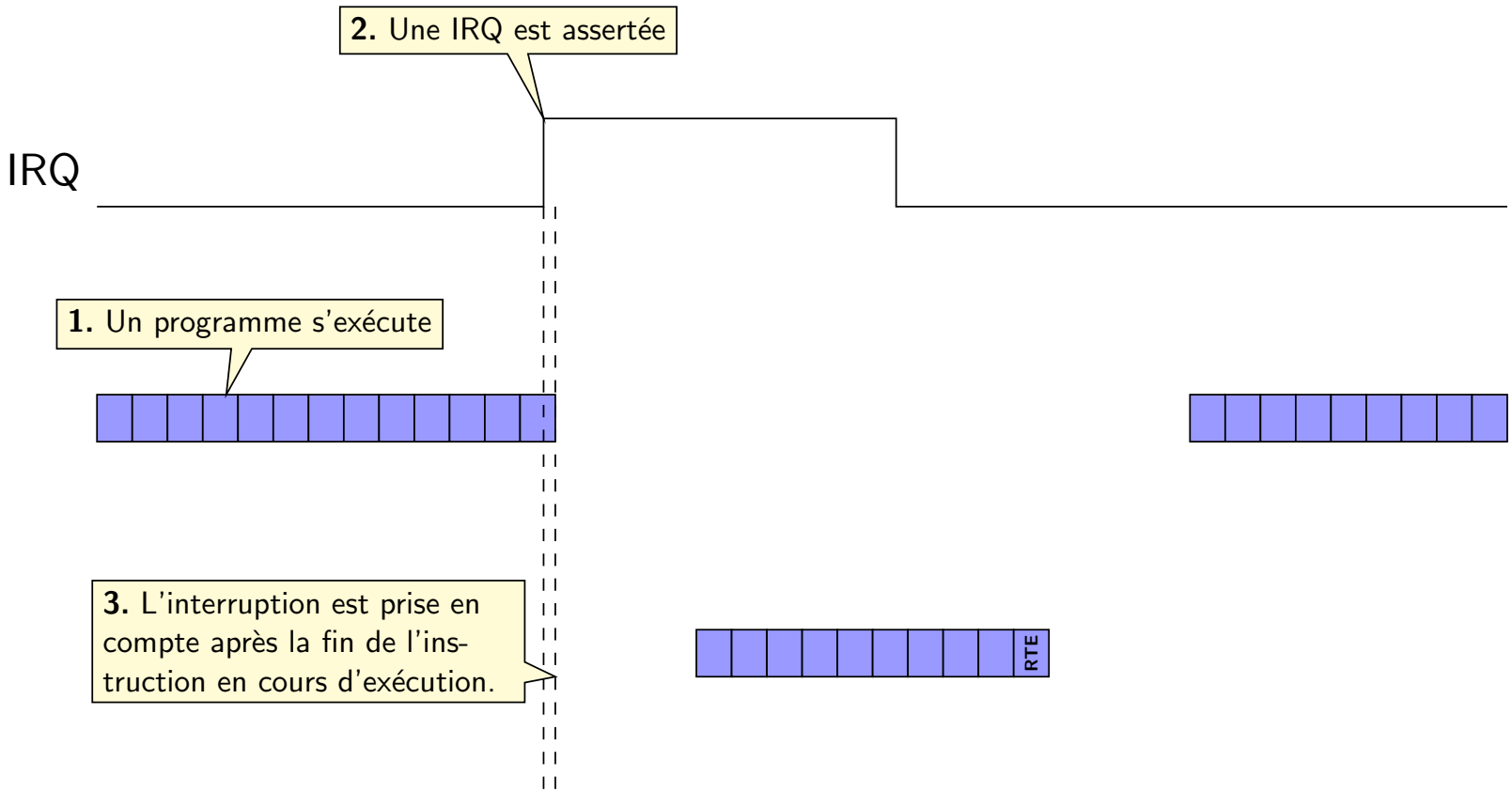
Interruptions (2)



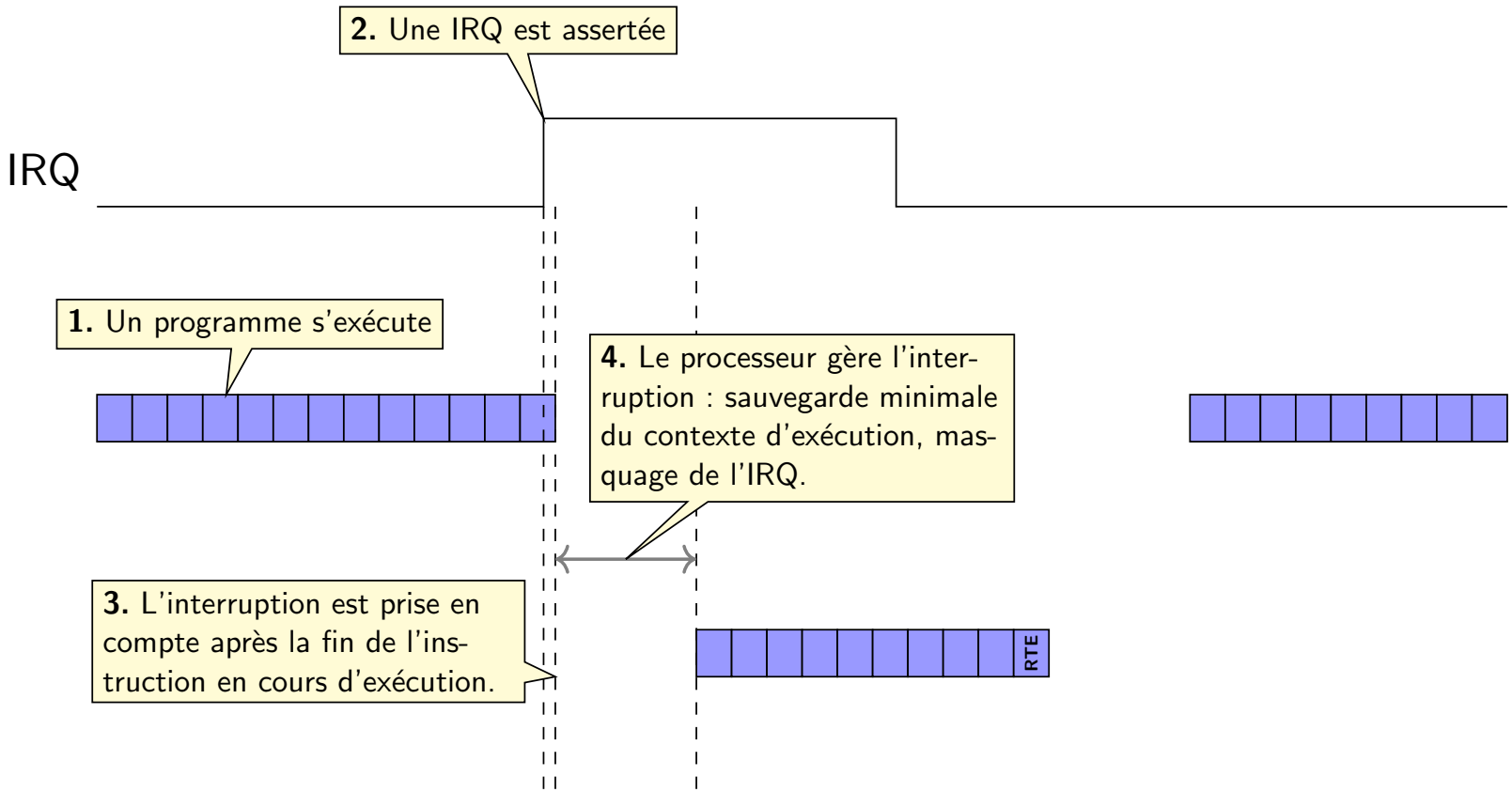
Interruptions (2)



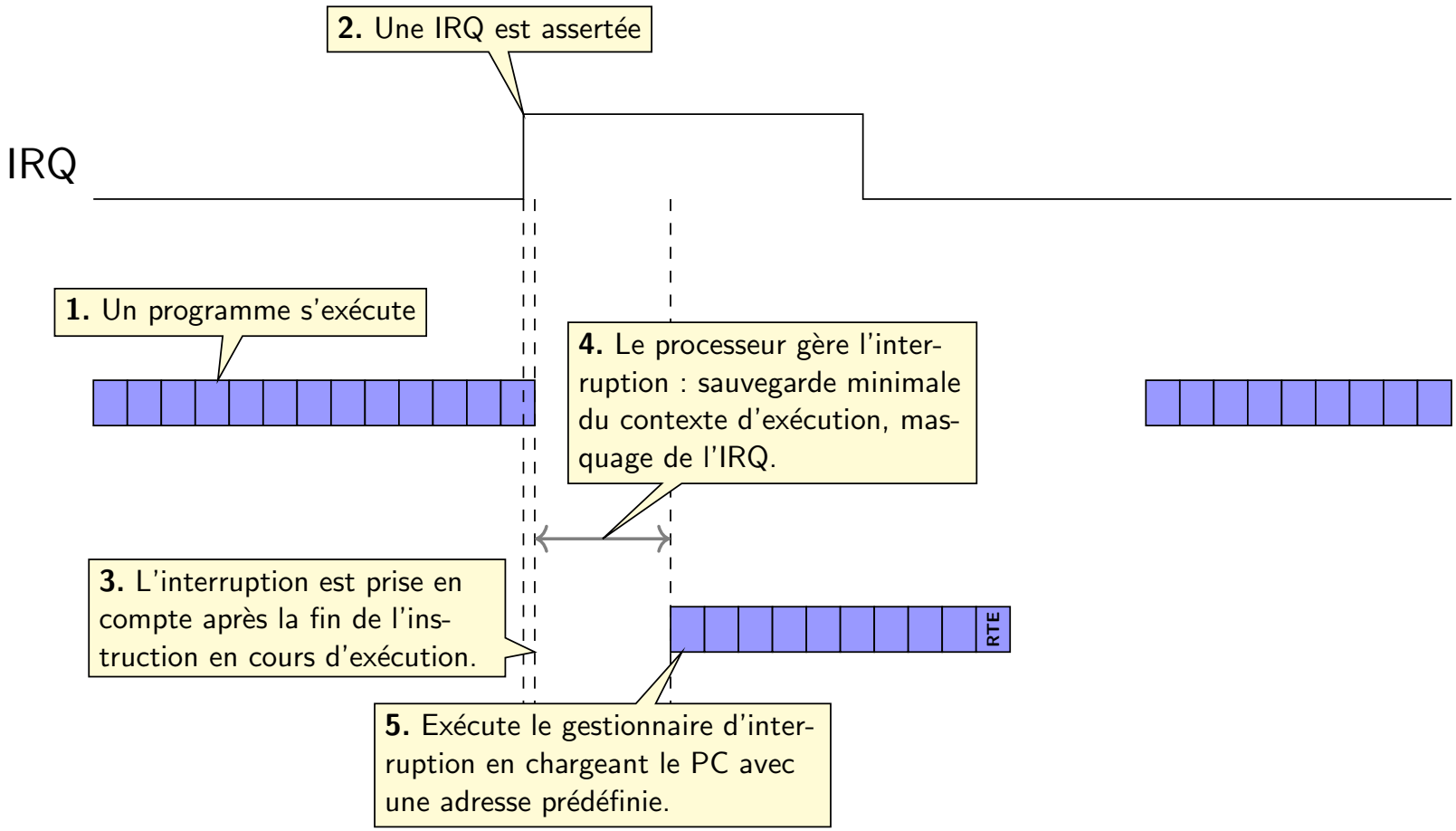
Interruptions (2)



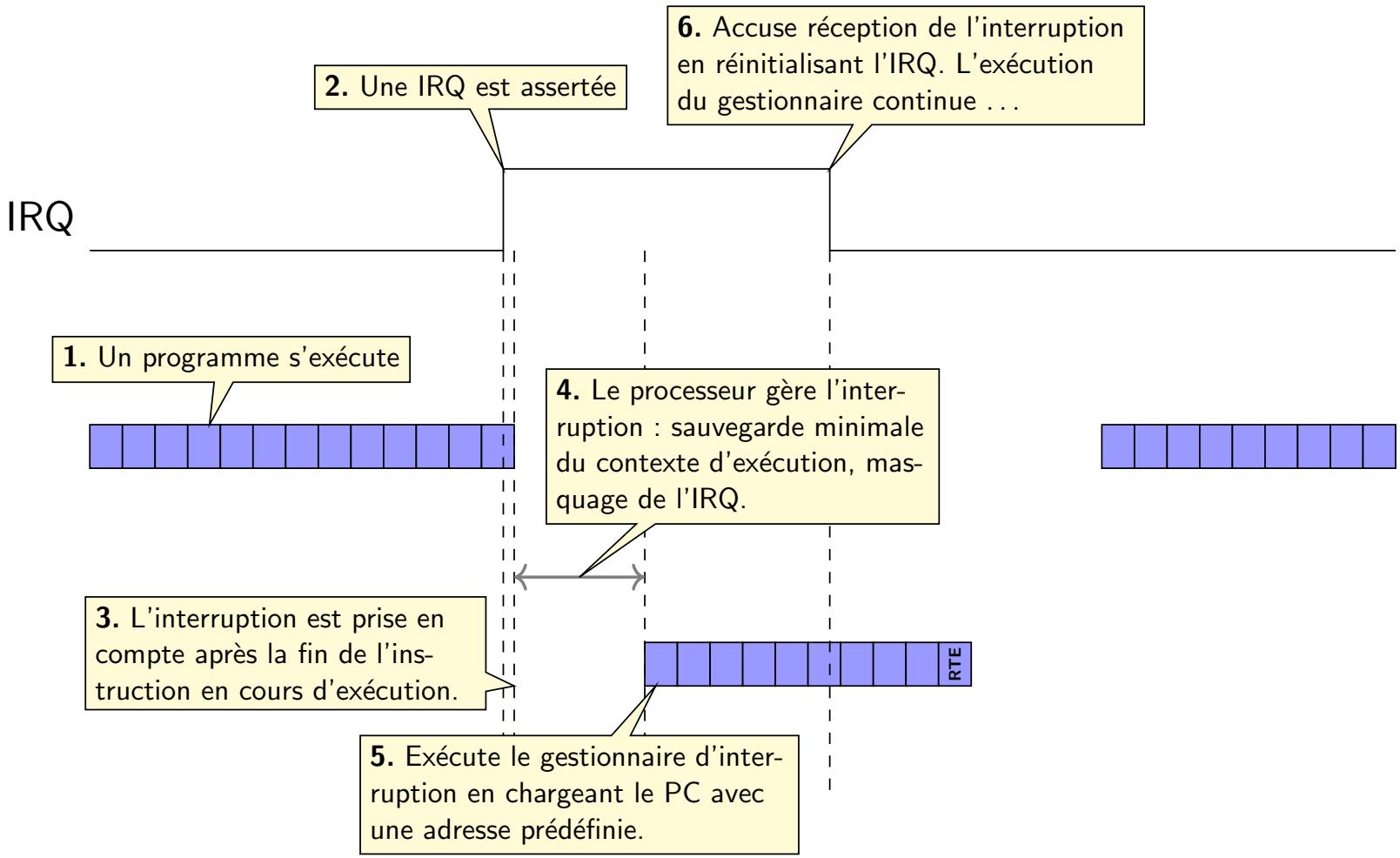
Interruptions (2)



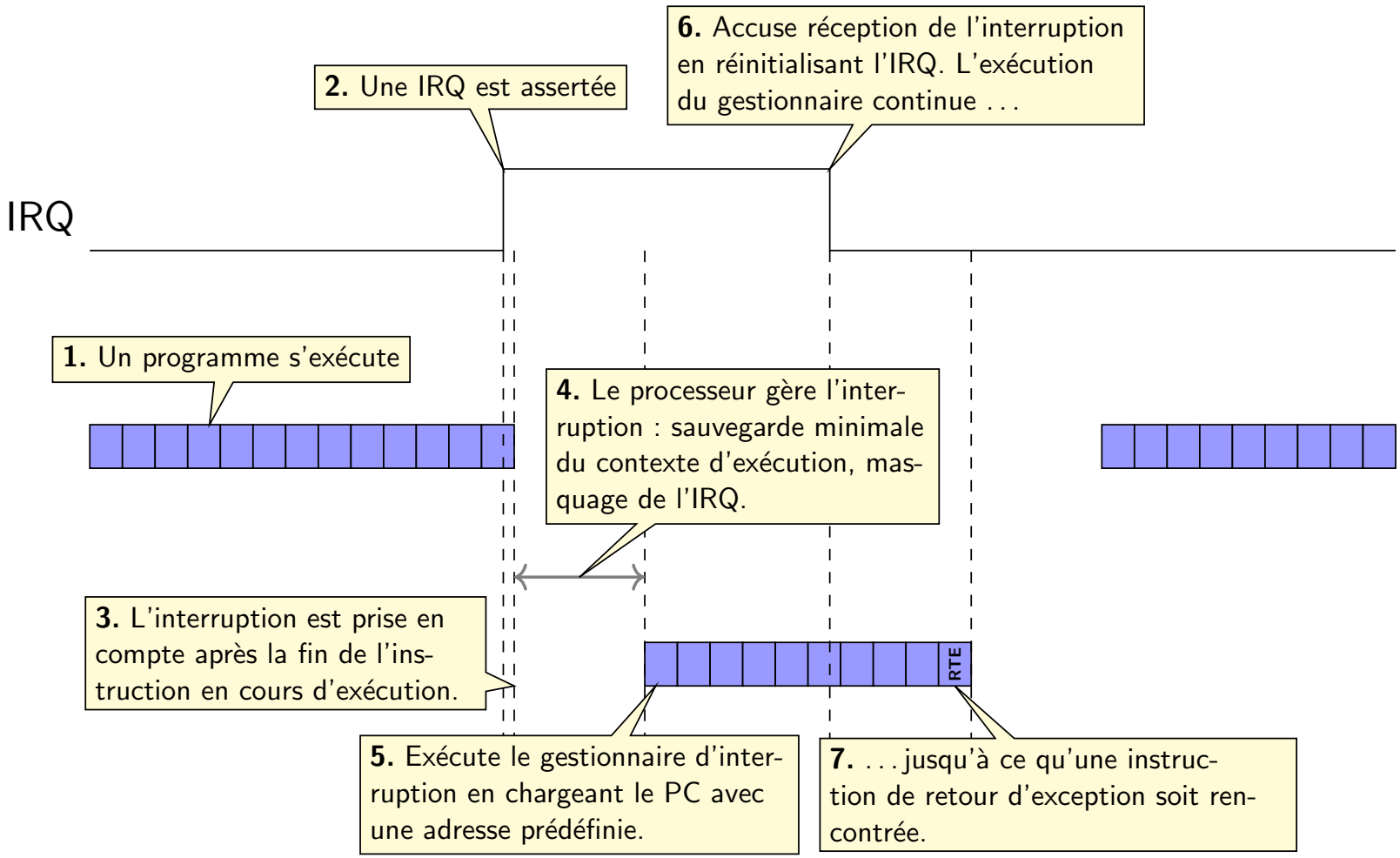
Interruptions (2)



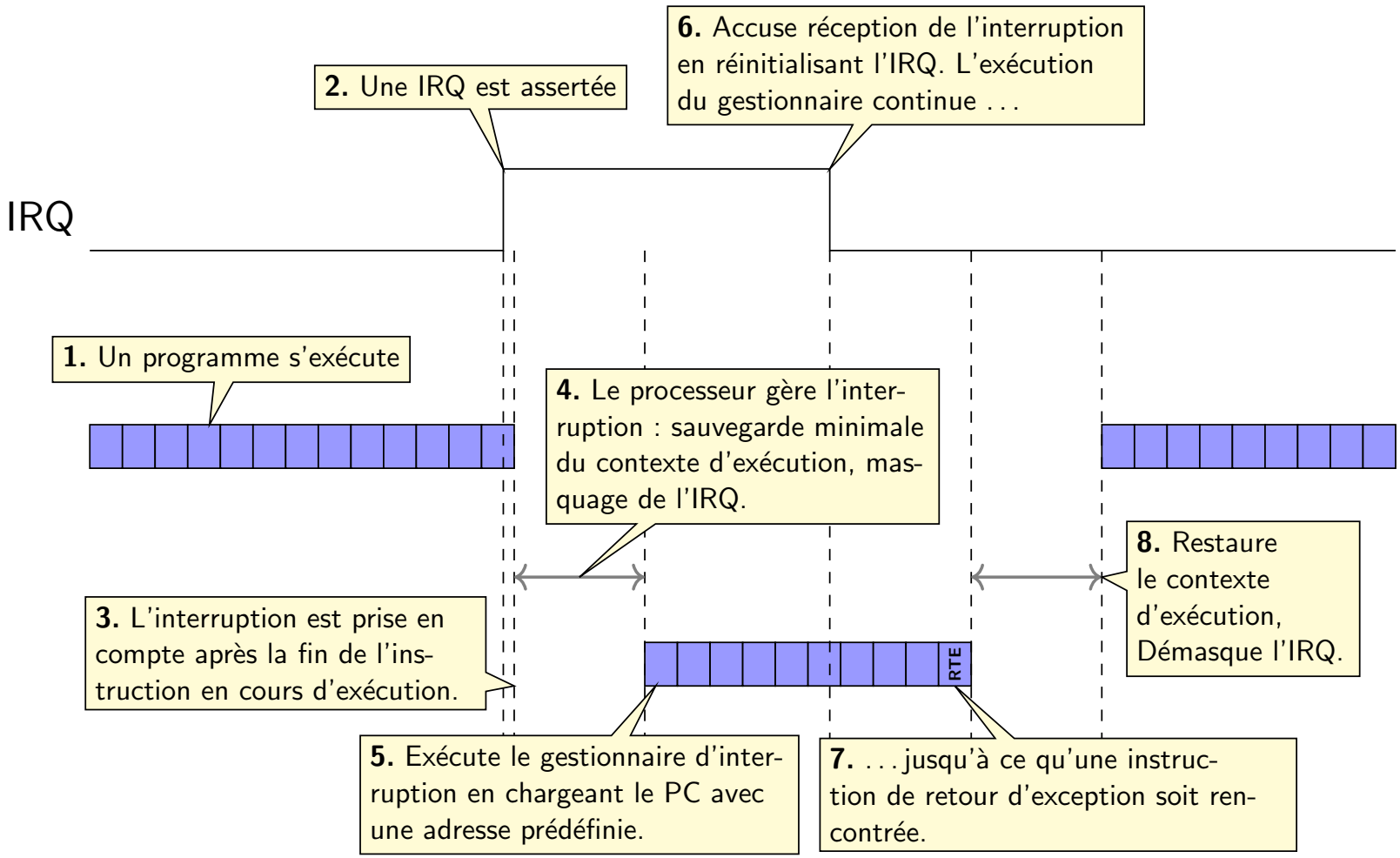
Interruptions (2)



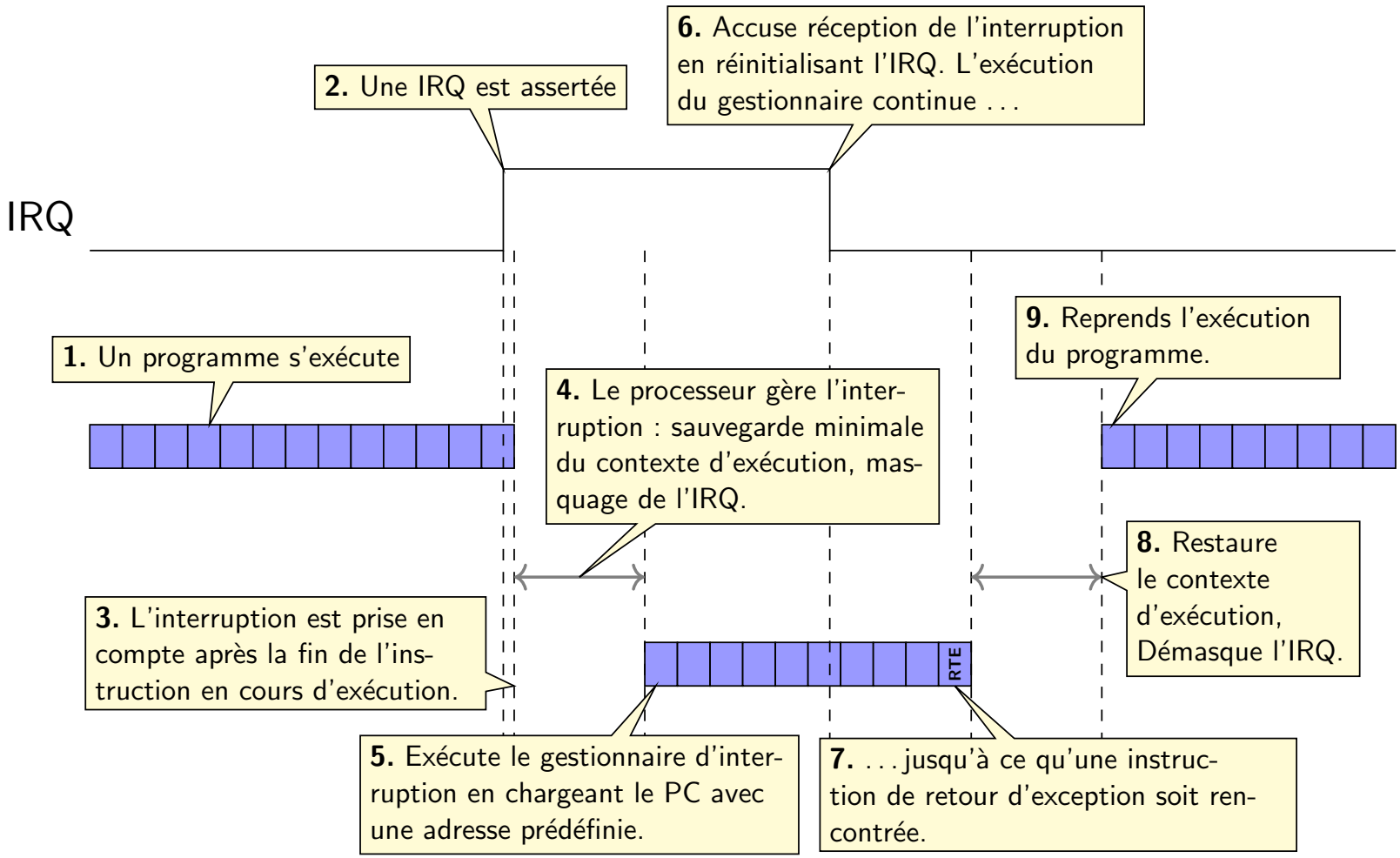
Interruptions (2)



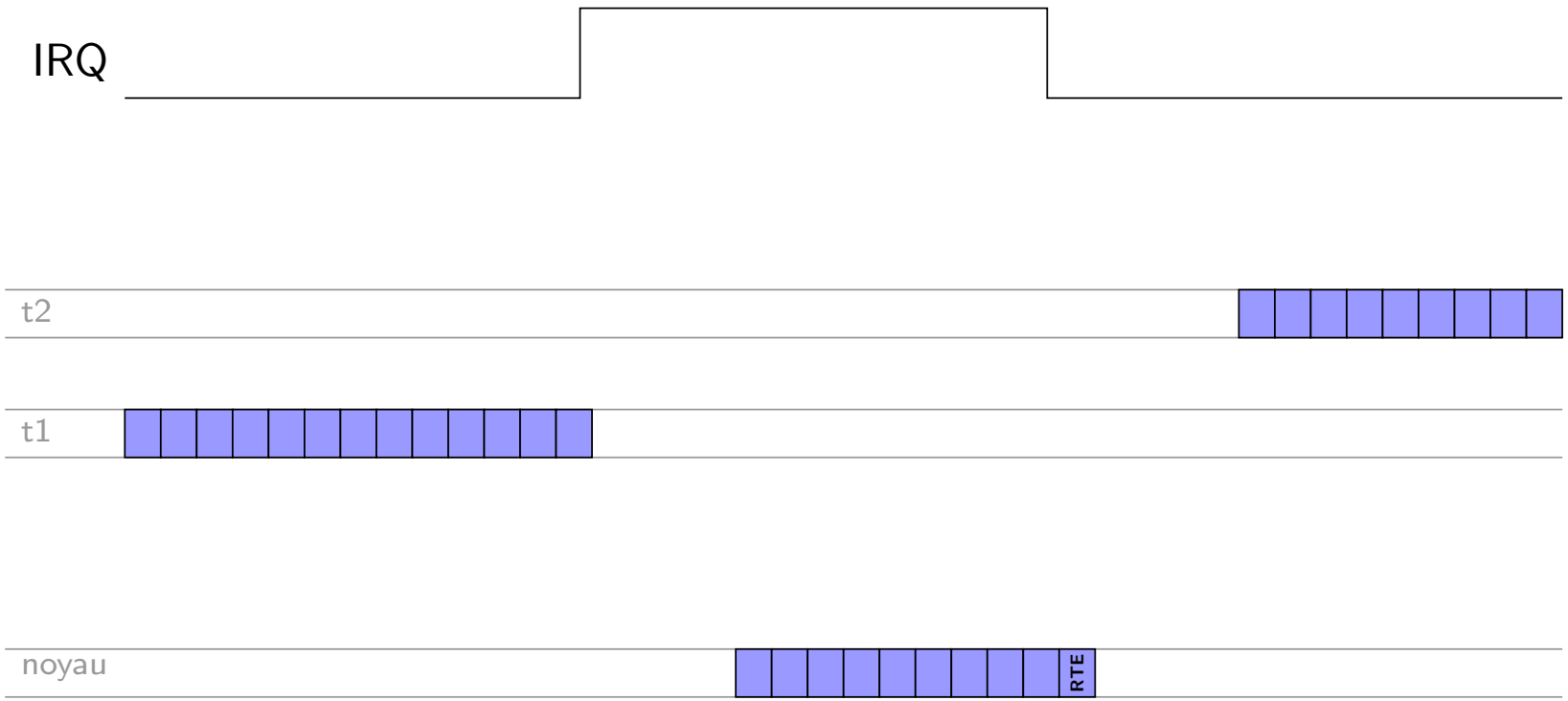
Interruptions (2)



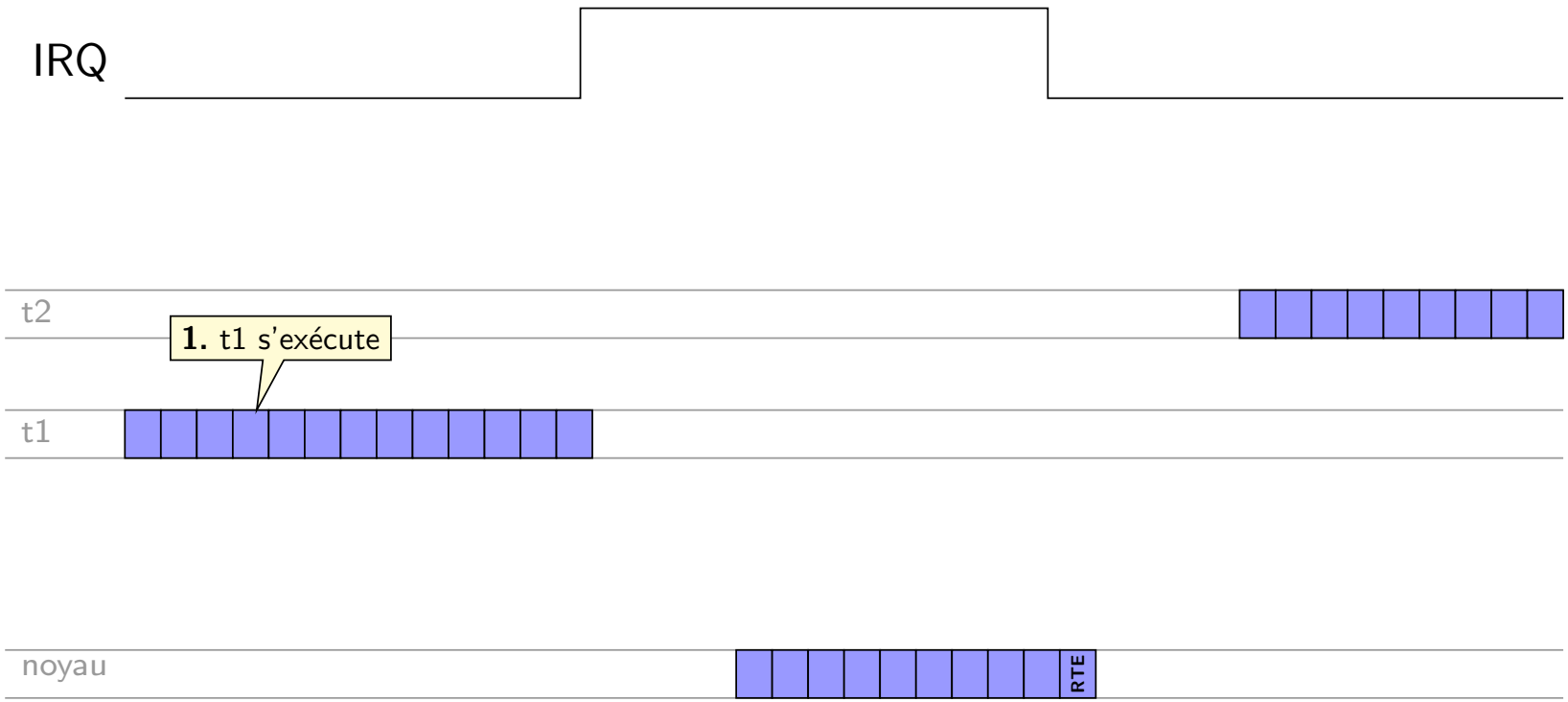
Interruptions (2)



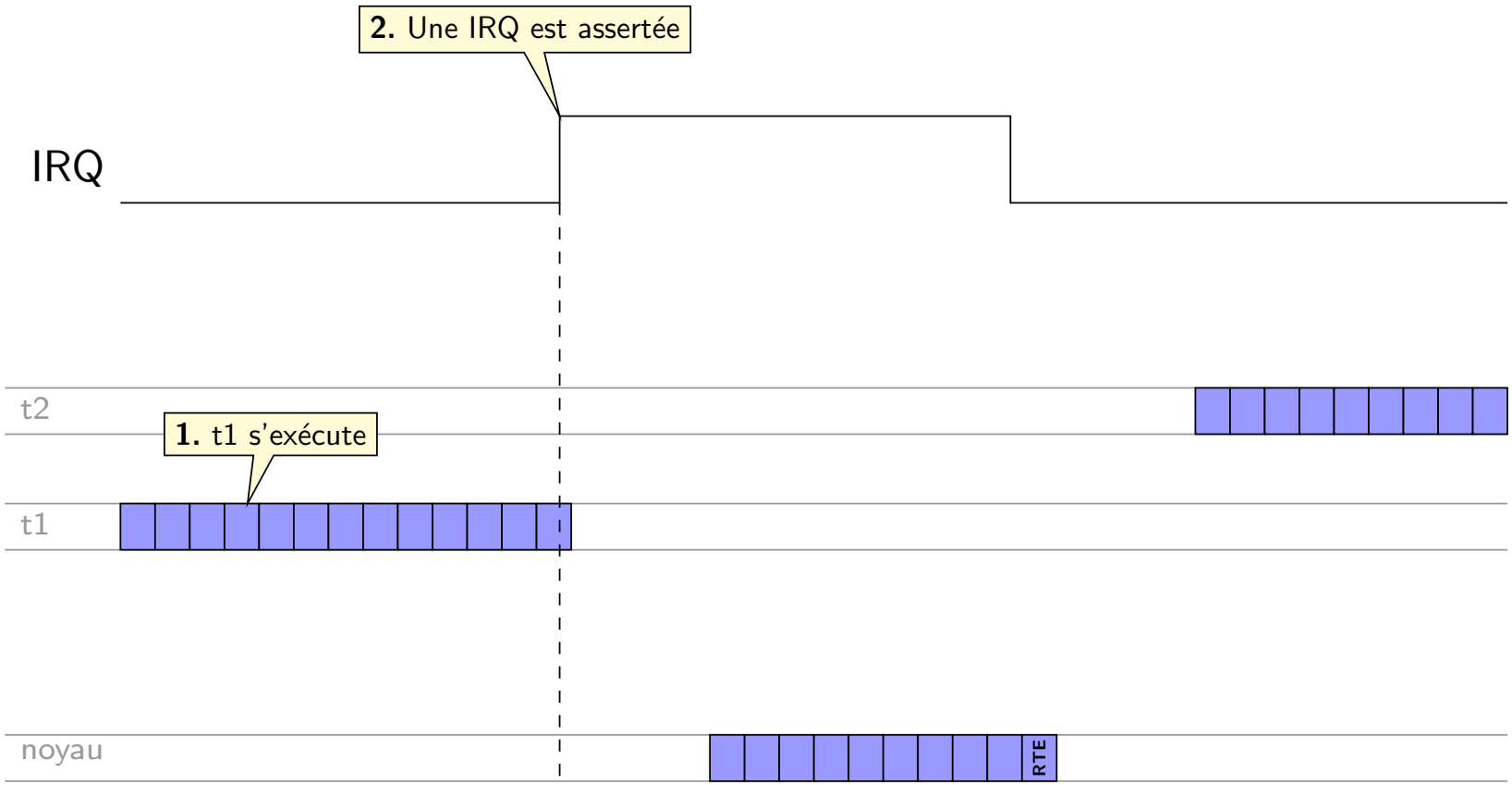
Interruptions et préemption



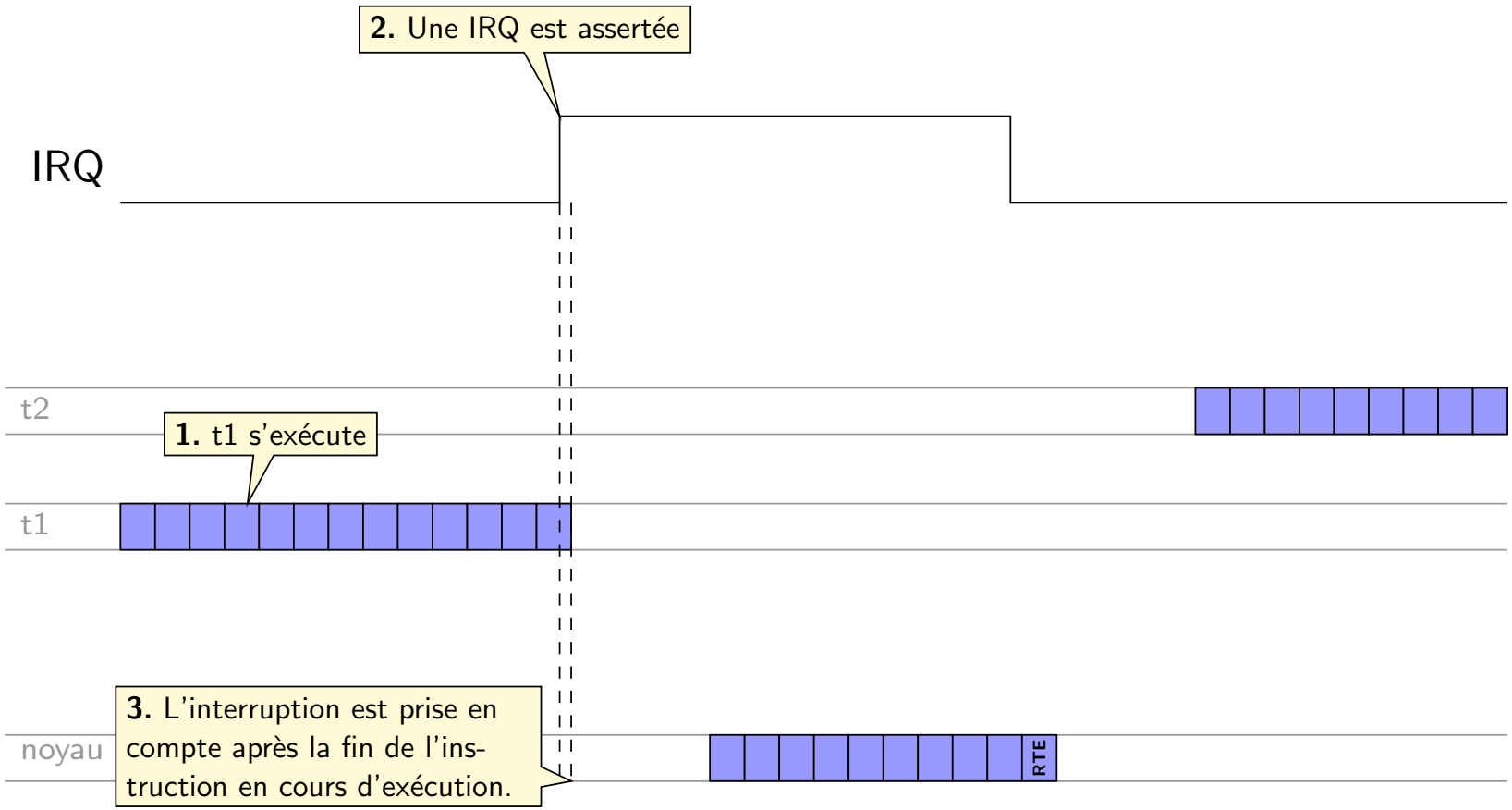
Interruptions et préemption



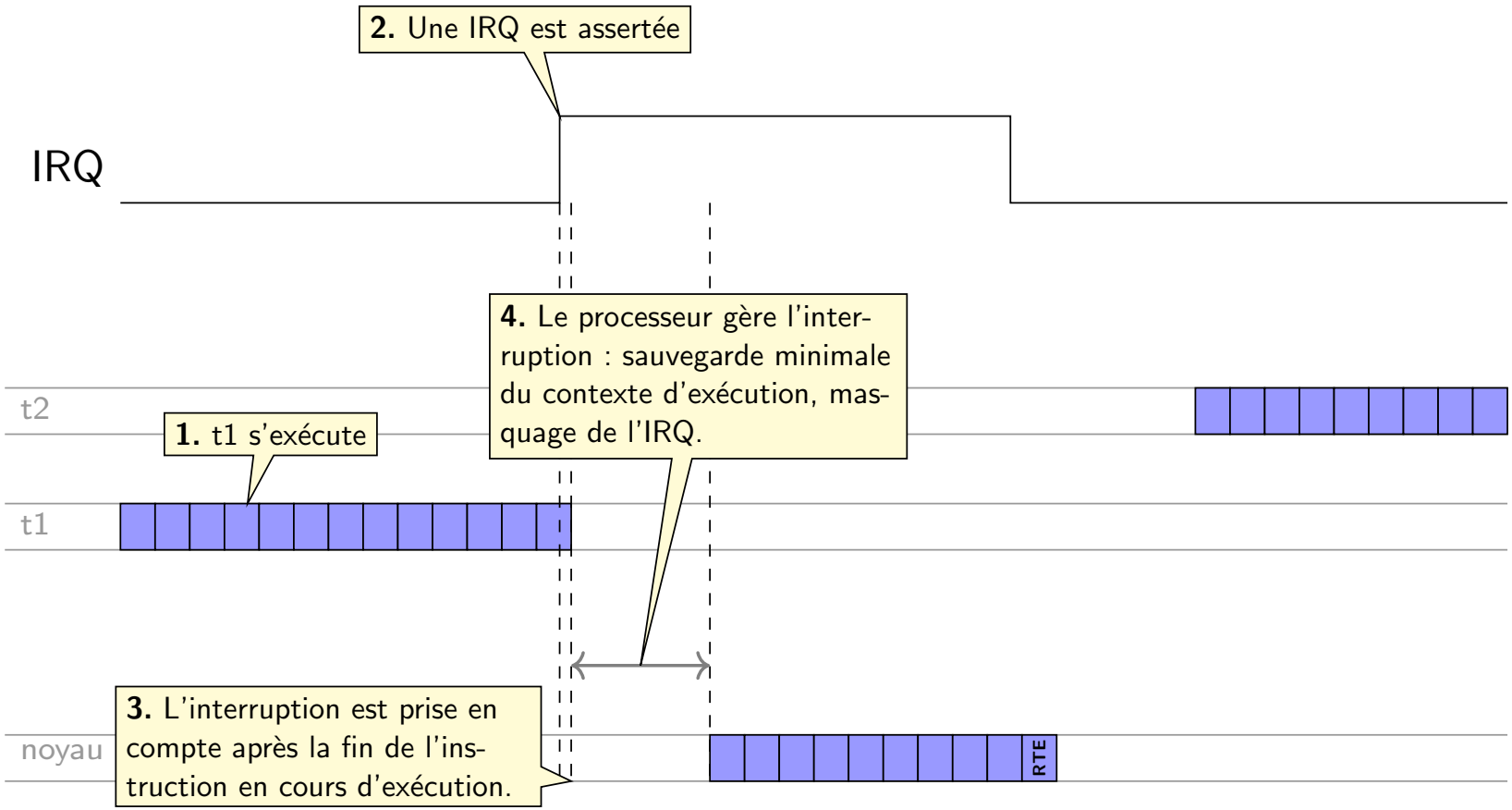
Interruptions et préemption



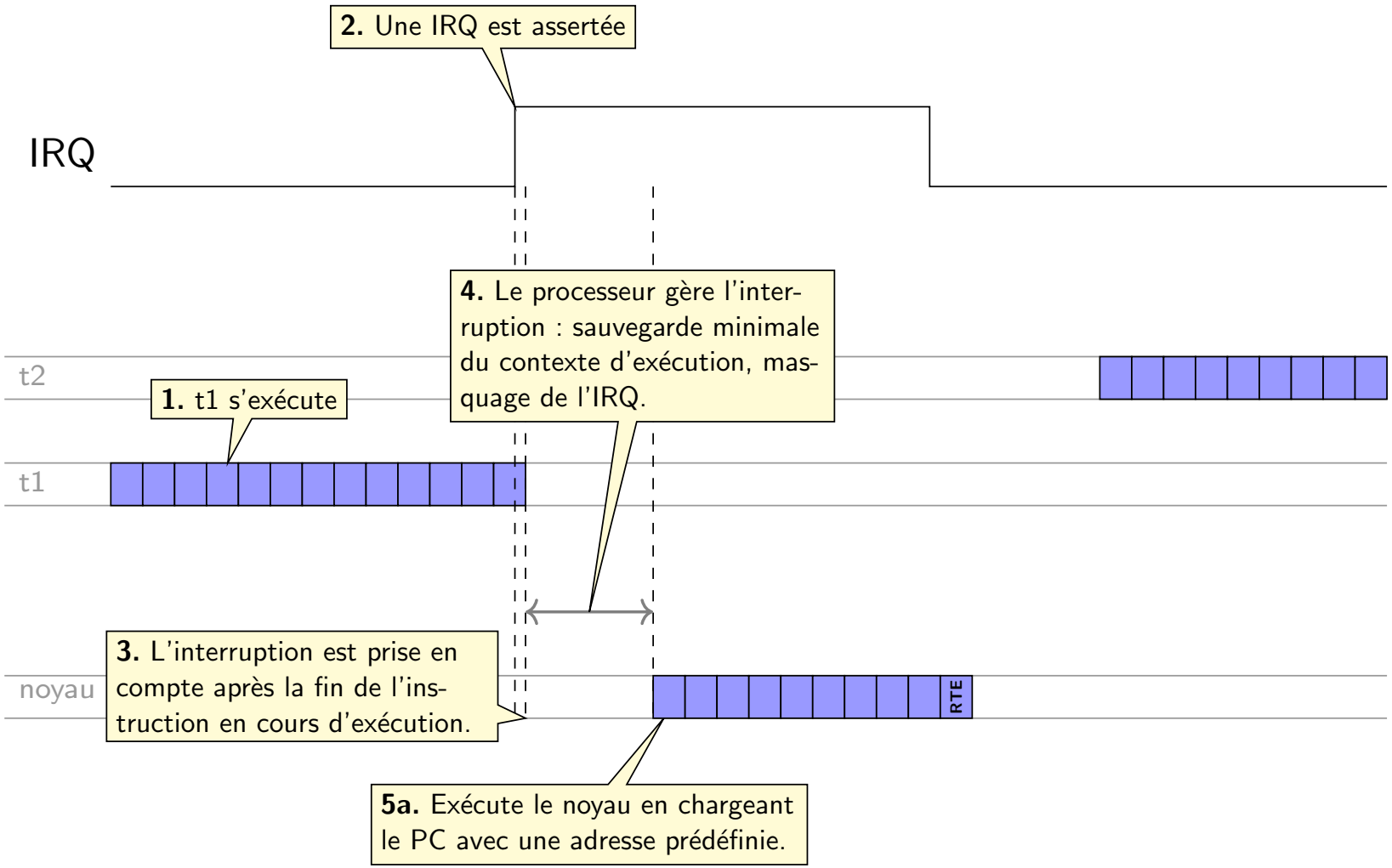
Interruptions et préemption



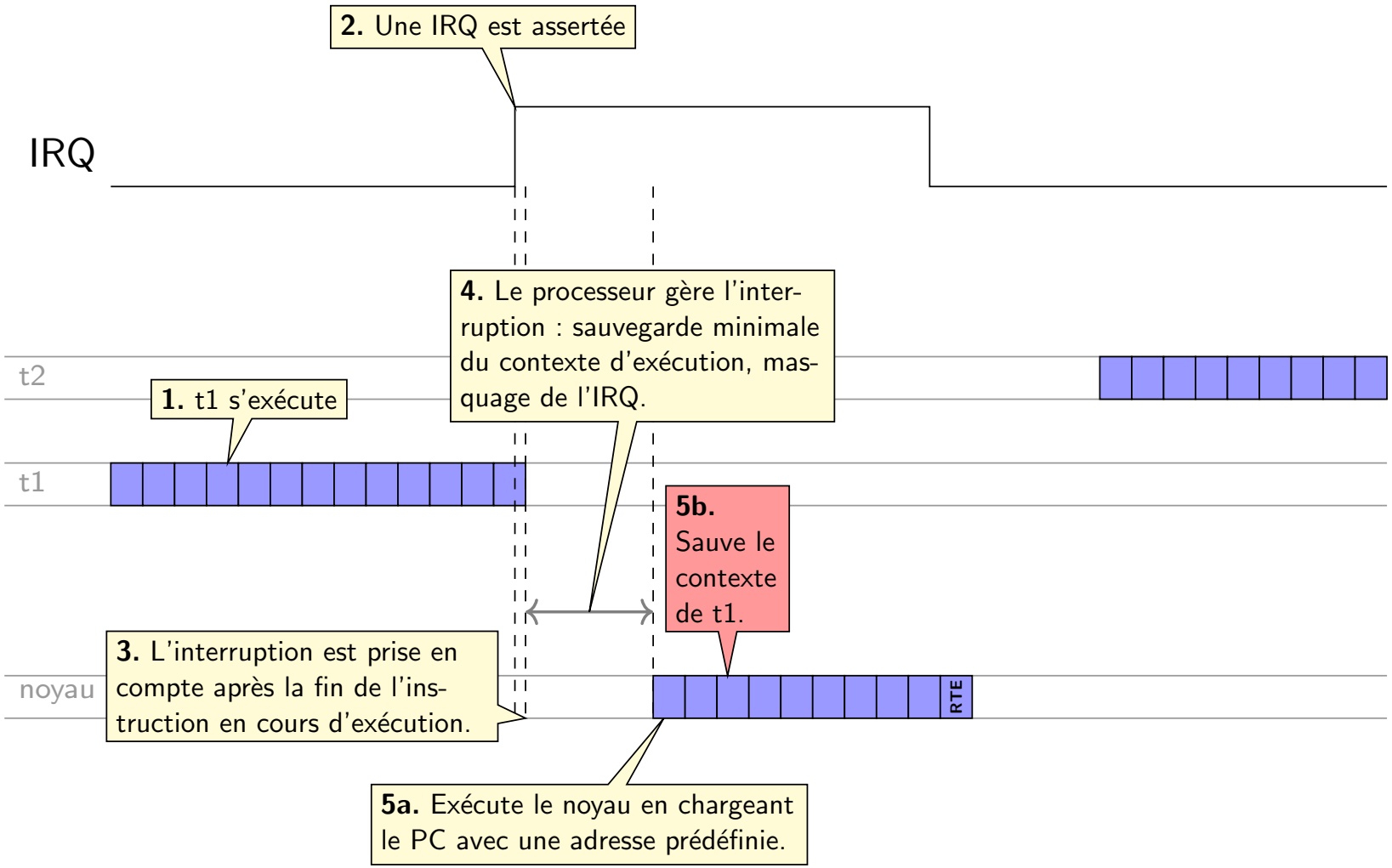
Interruptions et préemption



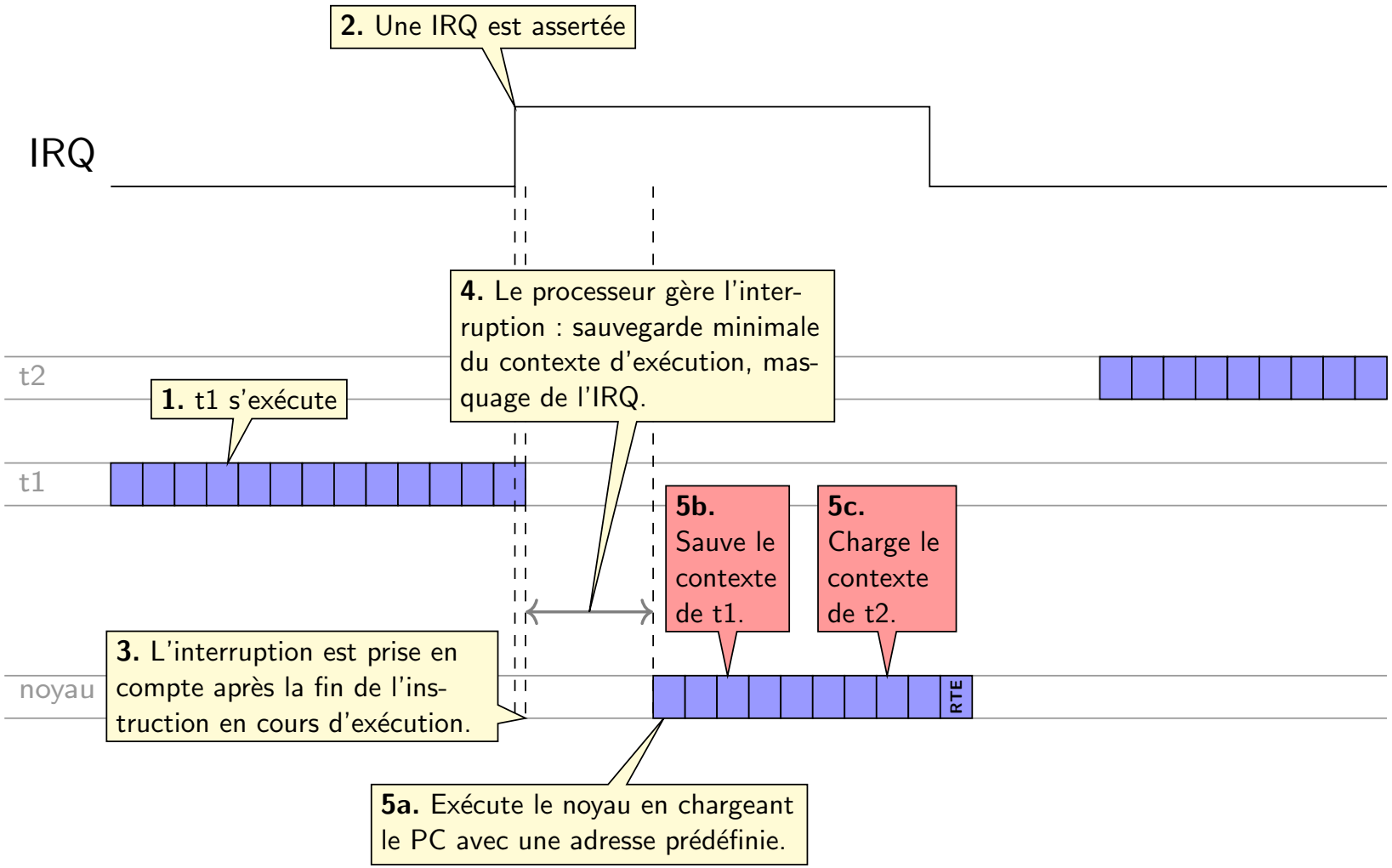
Interruptions et préemption



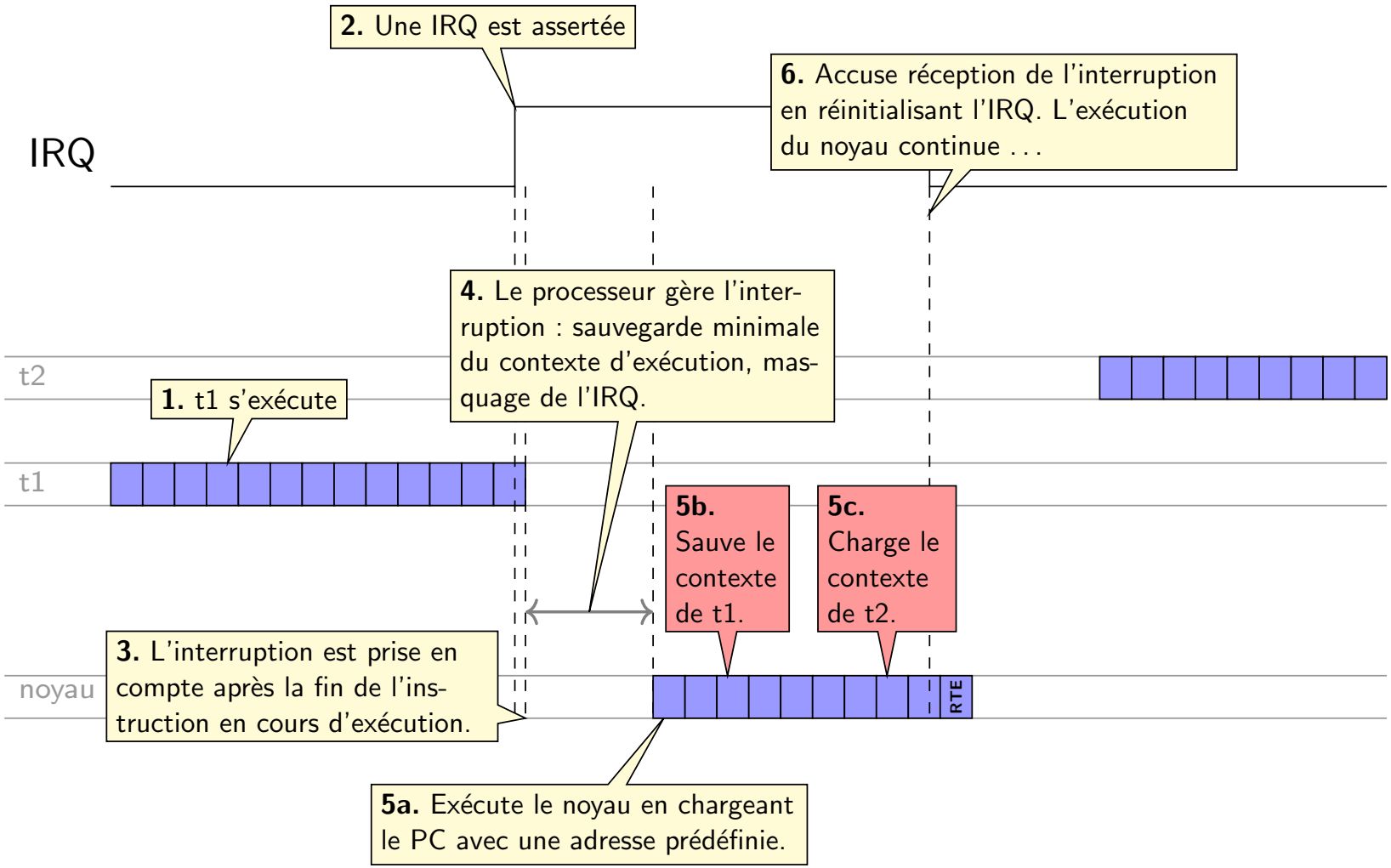
Interruptions et préemption



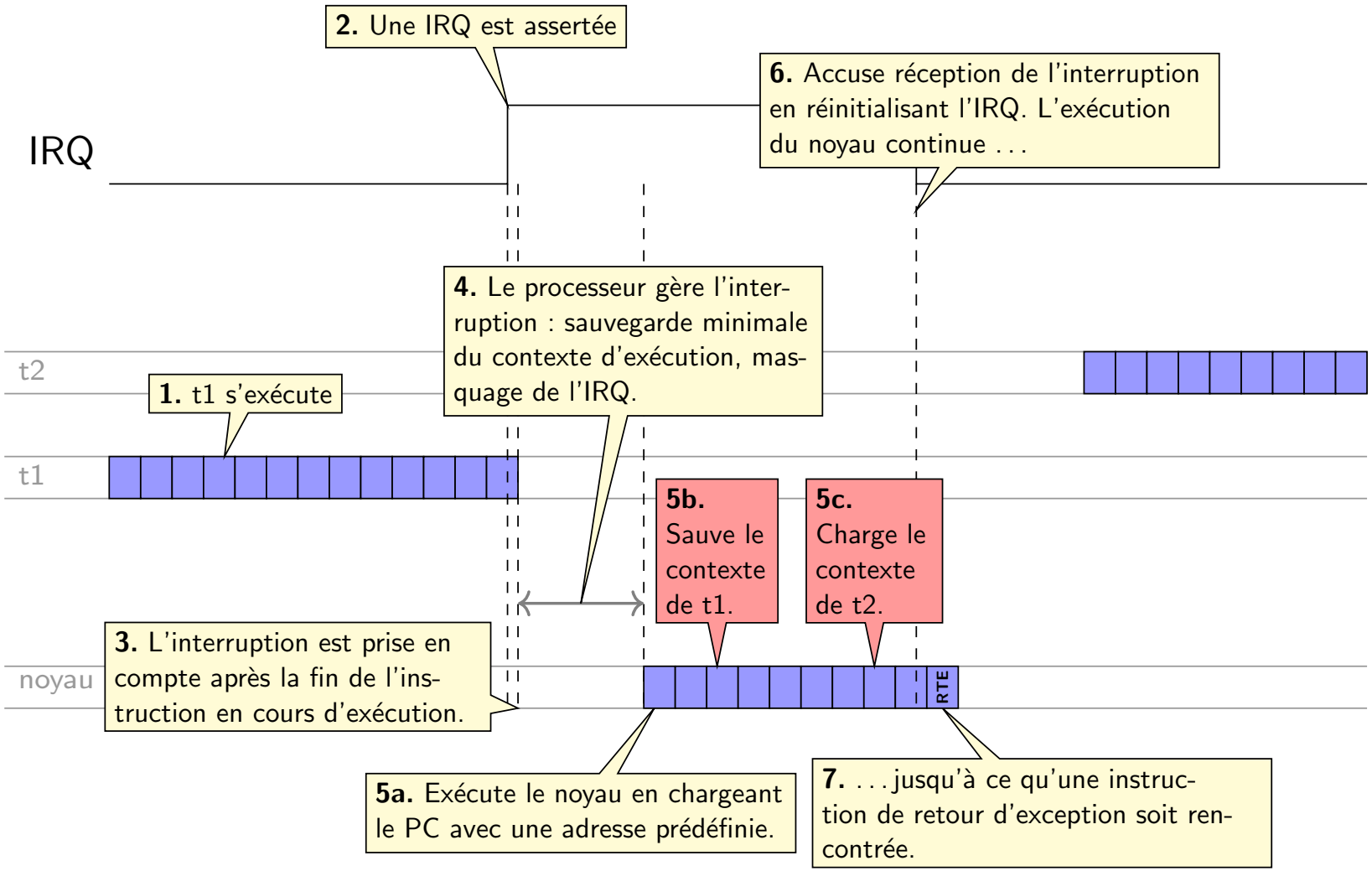
Interruptions et préemption



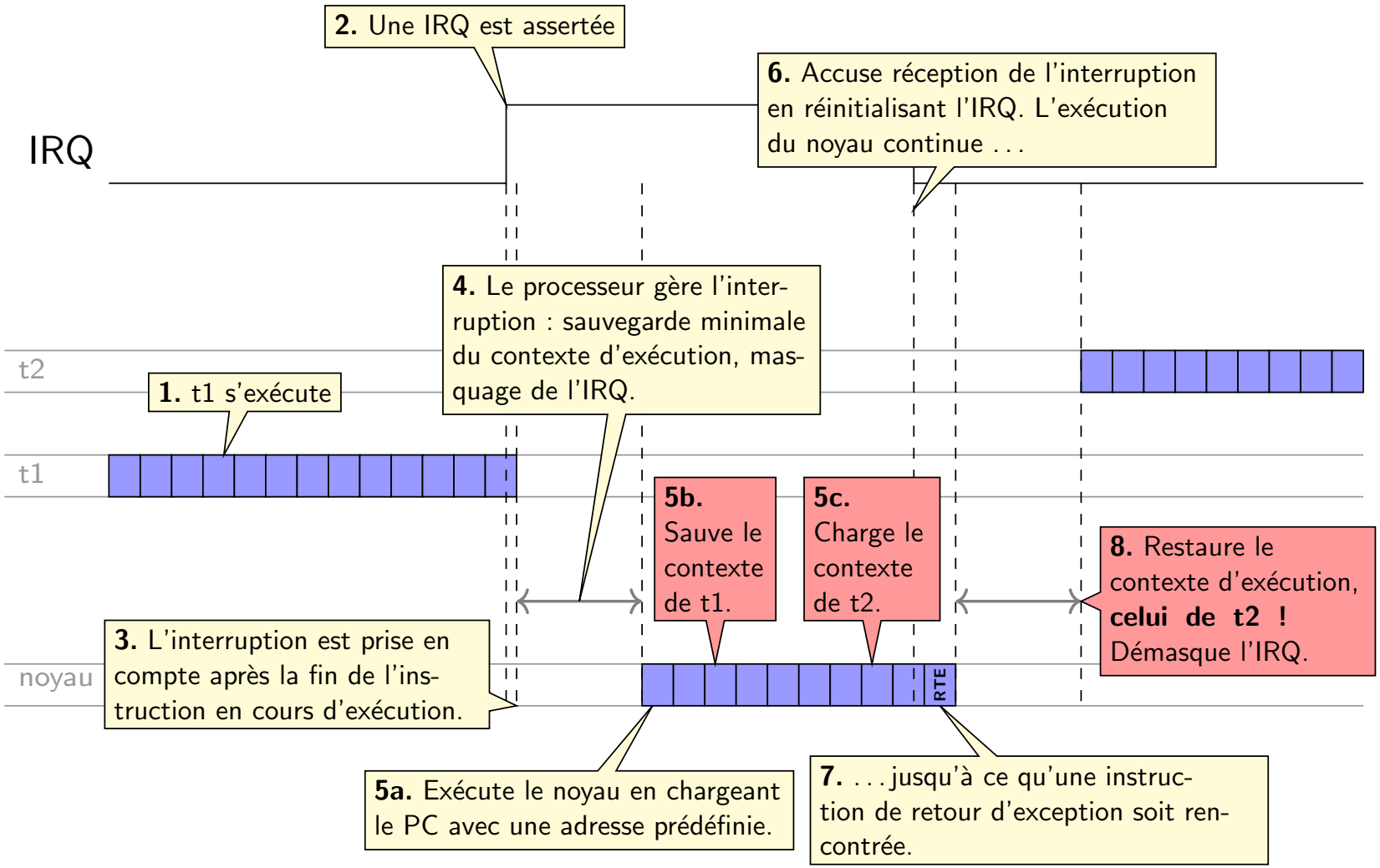
Interruptions et préemption



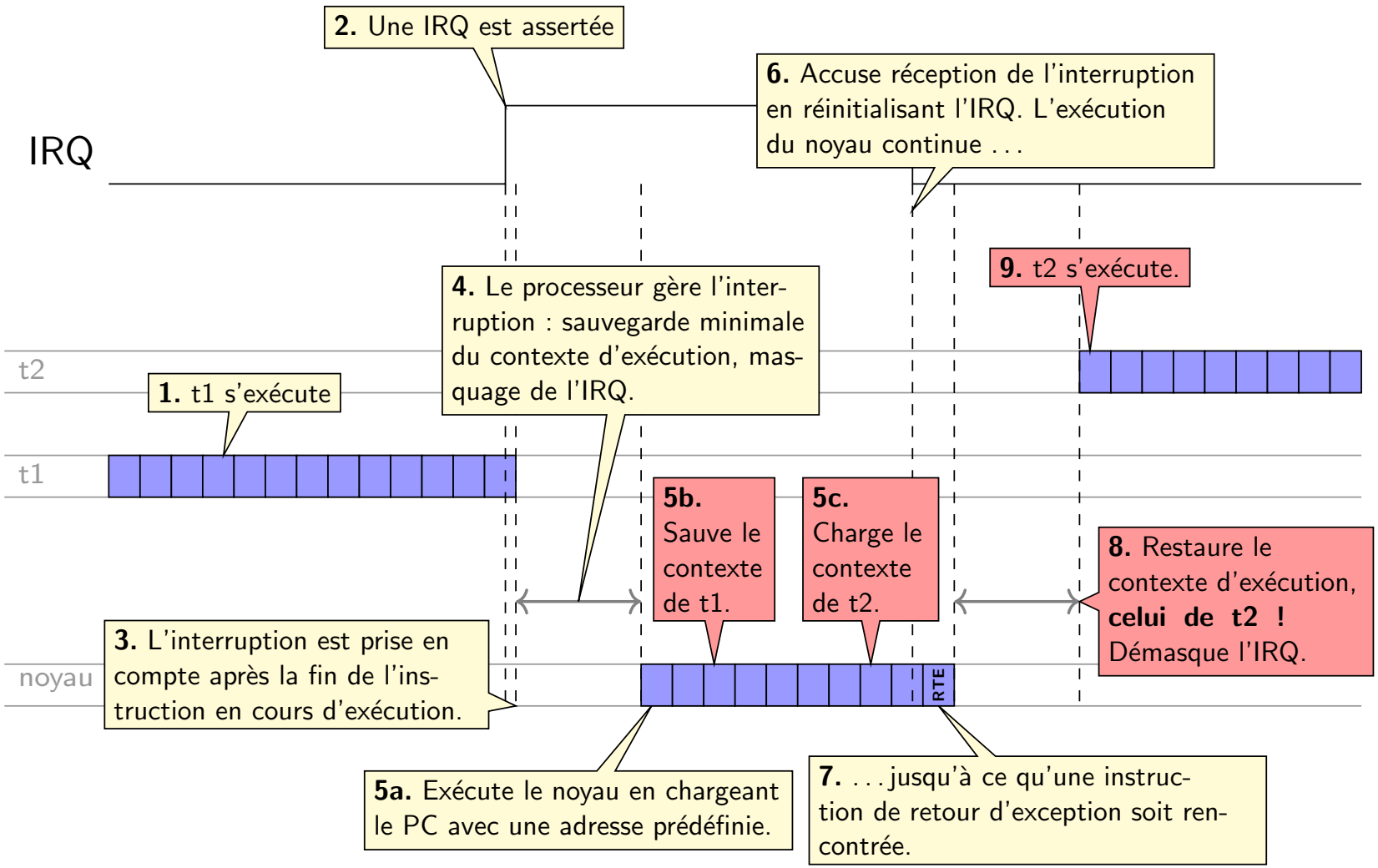
Interruptions et préemption



Interruptions et préemption

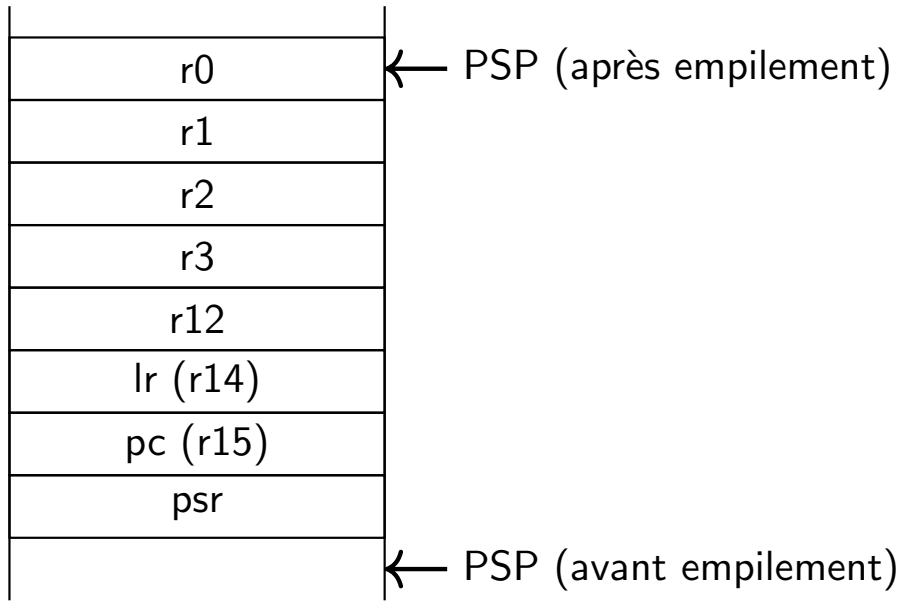


Interruptions et préemption



Changement de contexte, example du Cortex M (1)

- Lorsqu'une interruption (y compris logicielle) se produit, le MCU pousse 8 registres sur la pile :

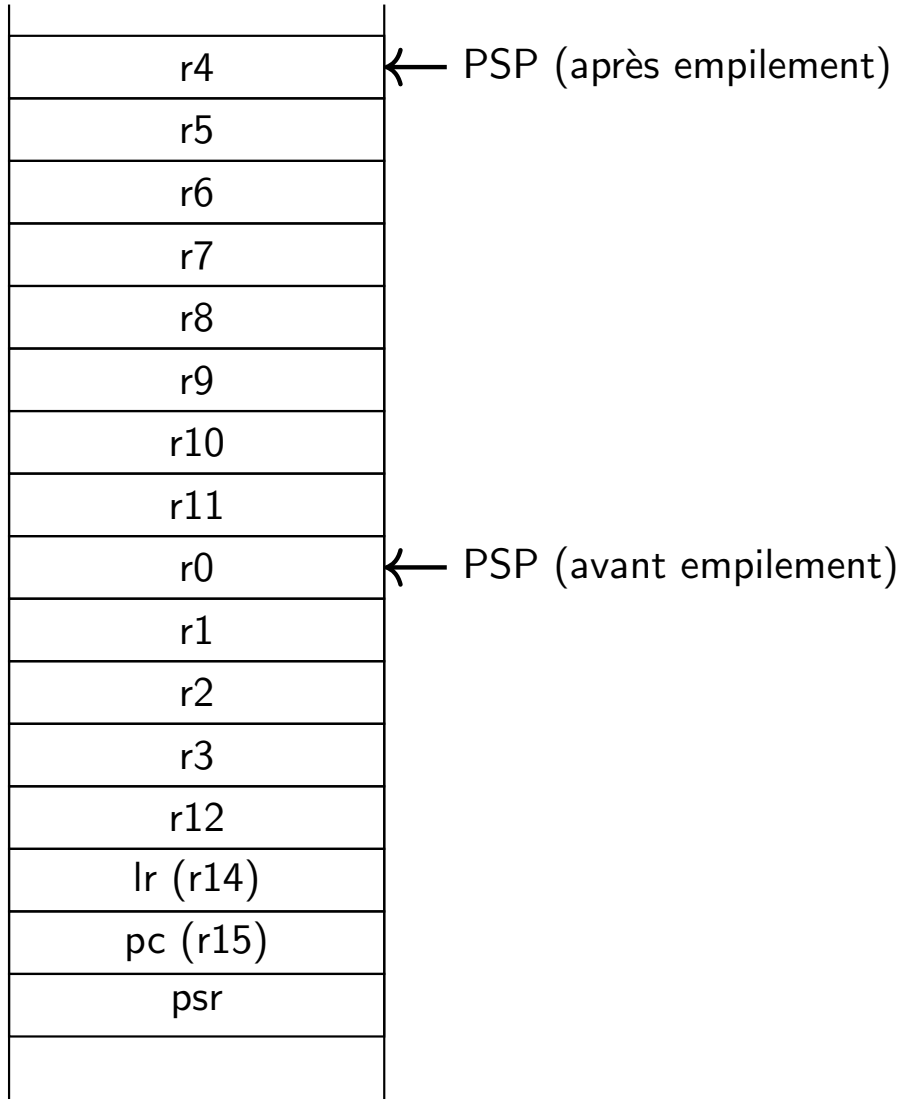


- Alors le MCU :
  1. Passe en mode superviseur
  2. Passe de la pile de processus (PSP) à la pile de handler (MSP),
  3. Exécute le handler correspondant.

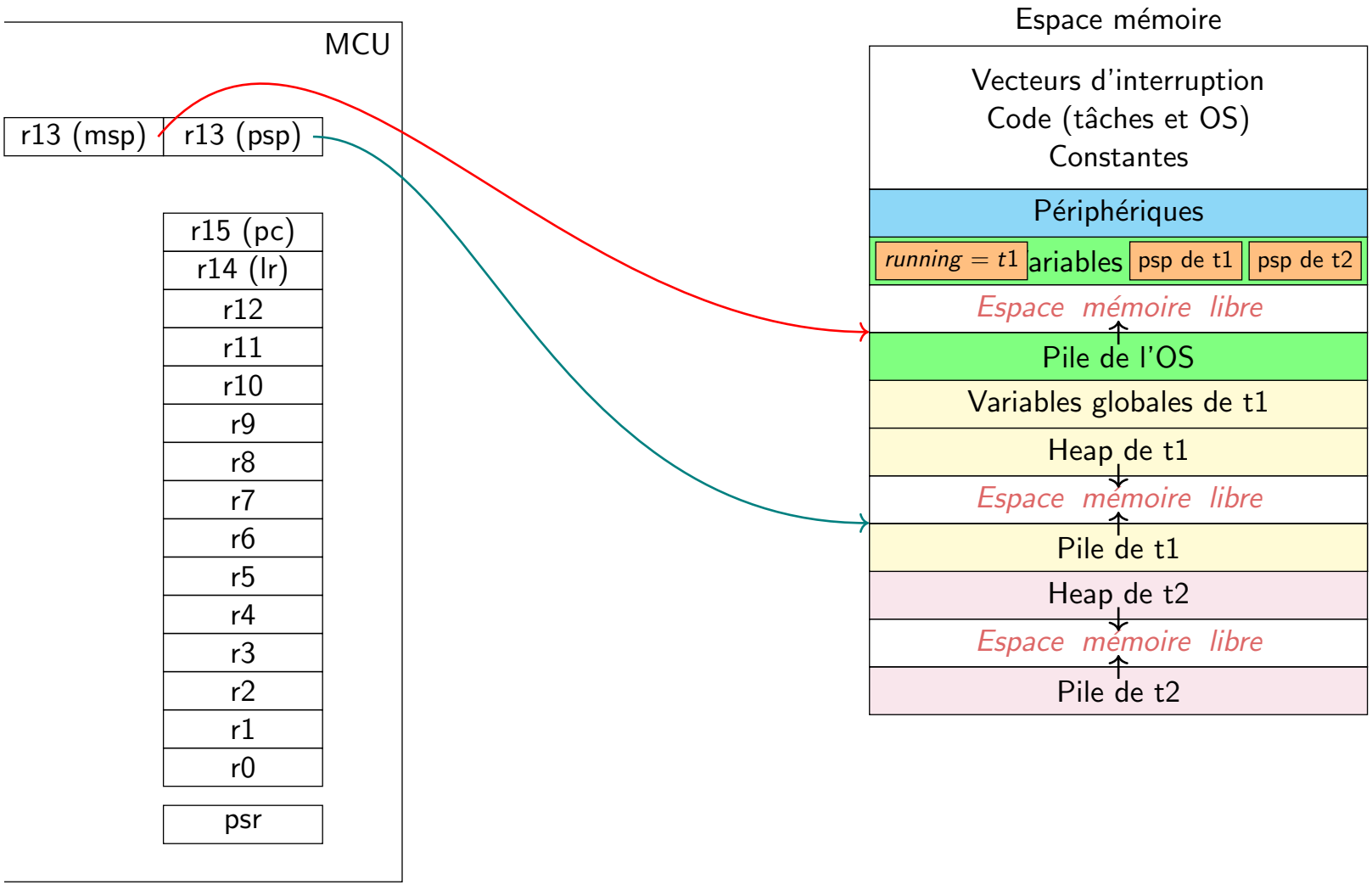
Changement de contexte, example du Cortex M (2)

Si un changement de contexte est nécessaire, le handler :

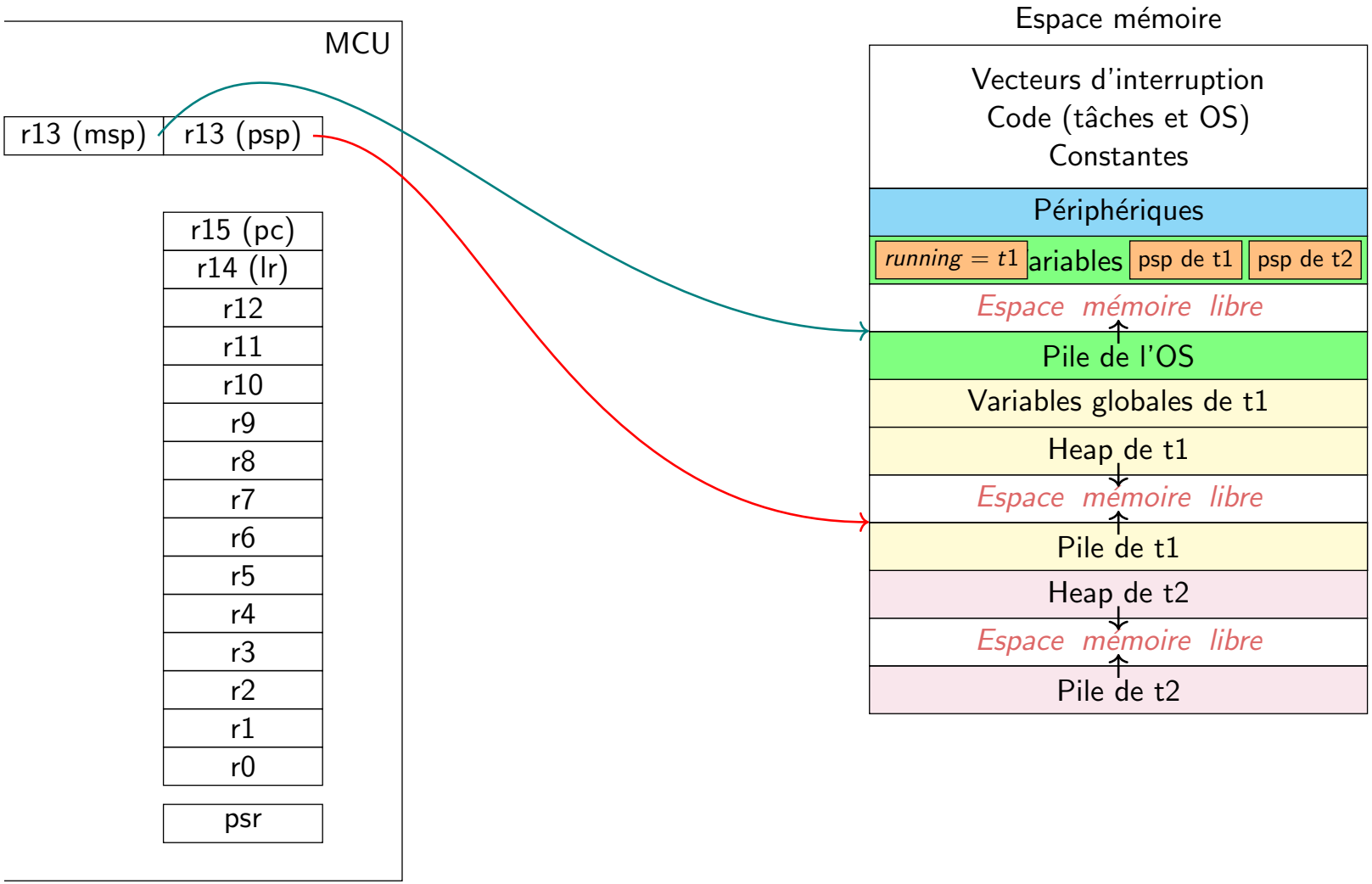
1. pousse les registres restants, sauf le PSP
2. stocke le PSP dans un champ du descripteur de tâche



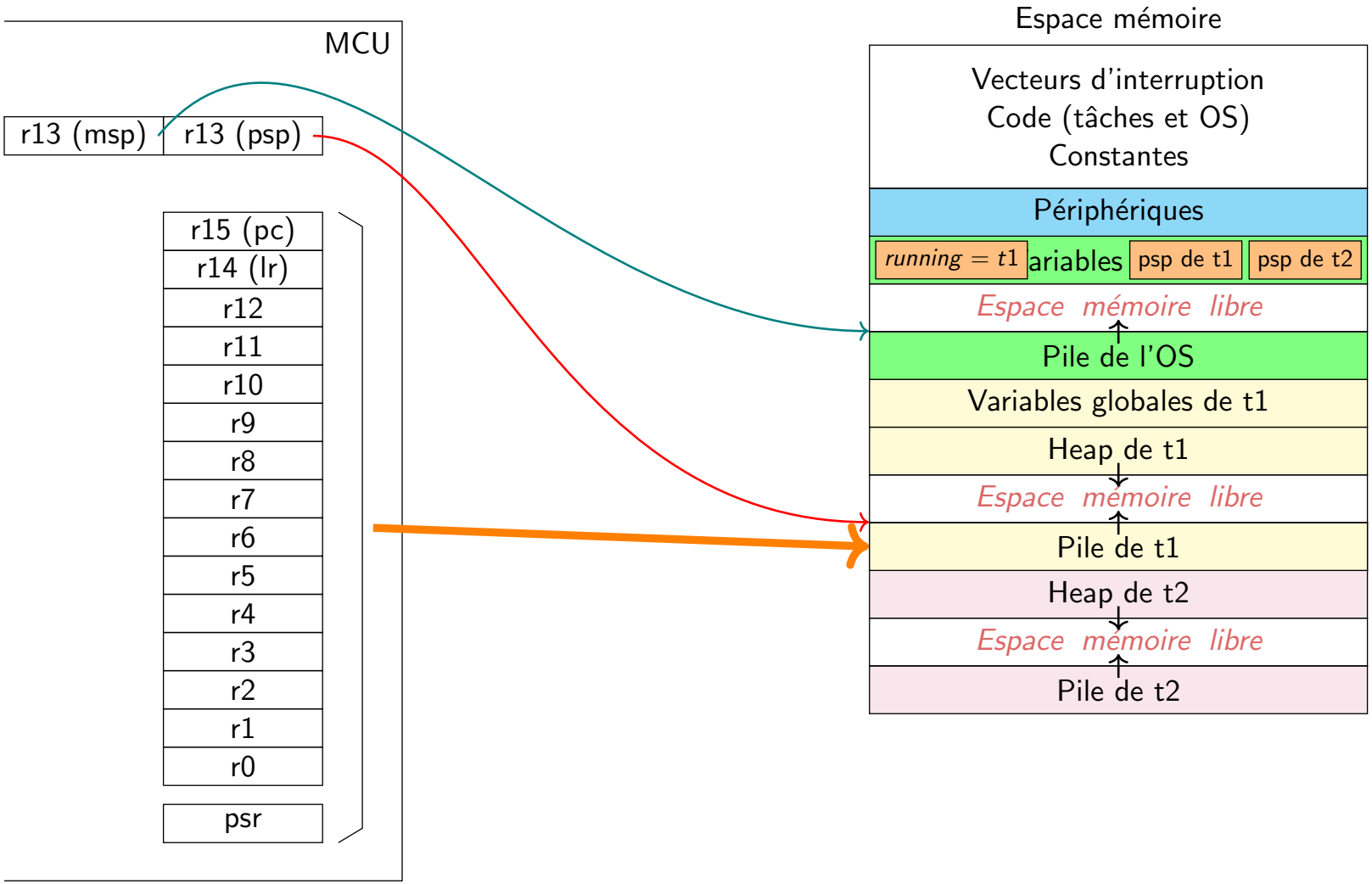
Changement de contexte, example du Cortex M (3)



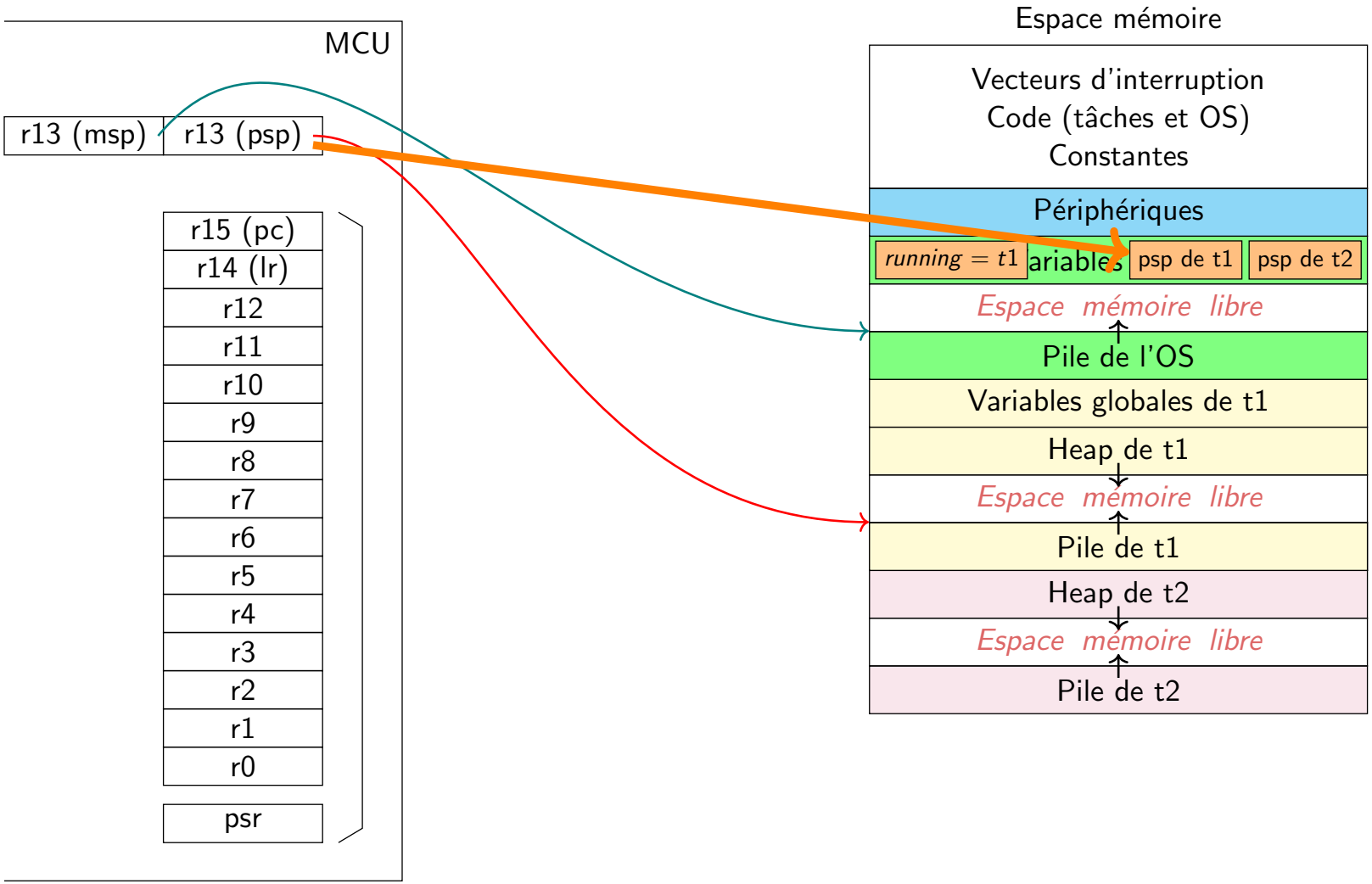
Changement de contexte, example du Cortex M (3)



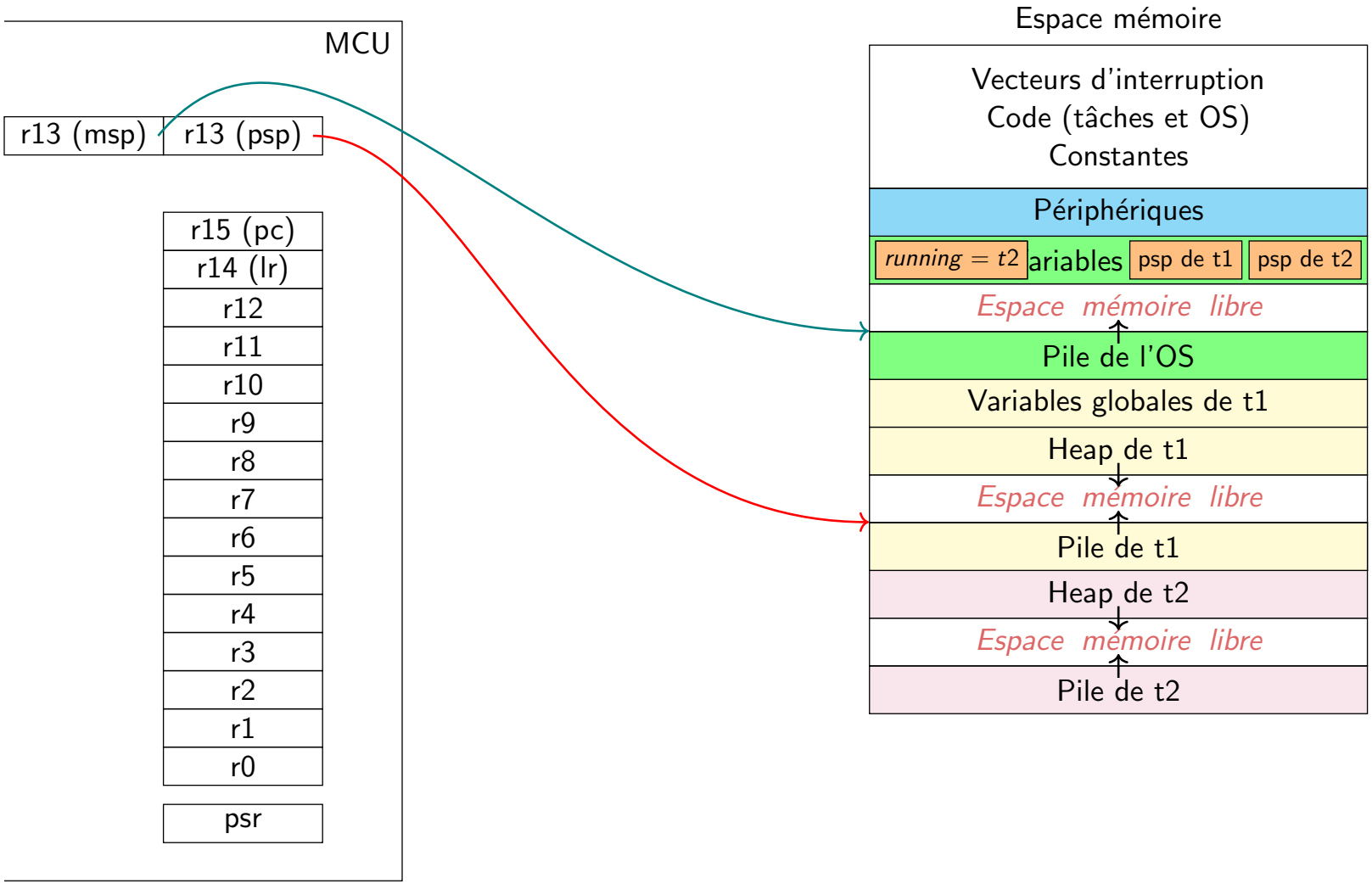
Changement de contexte, example du Cortex M (3)



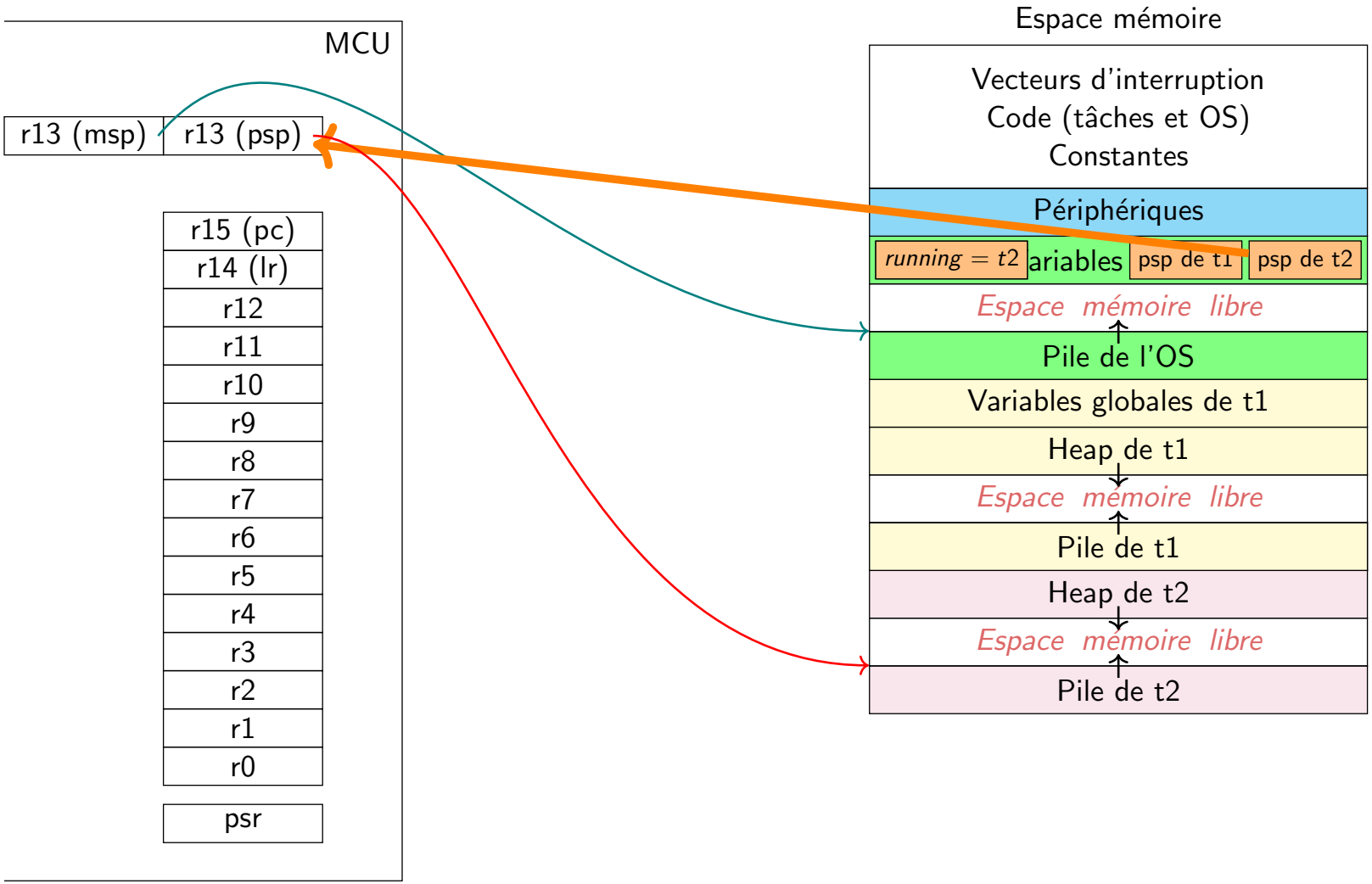
Changement de contexte, example du Cortex M (3)



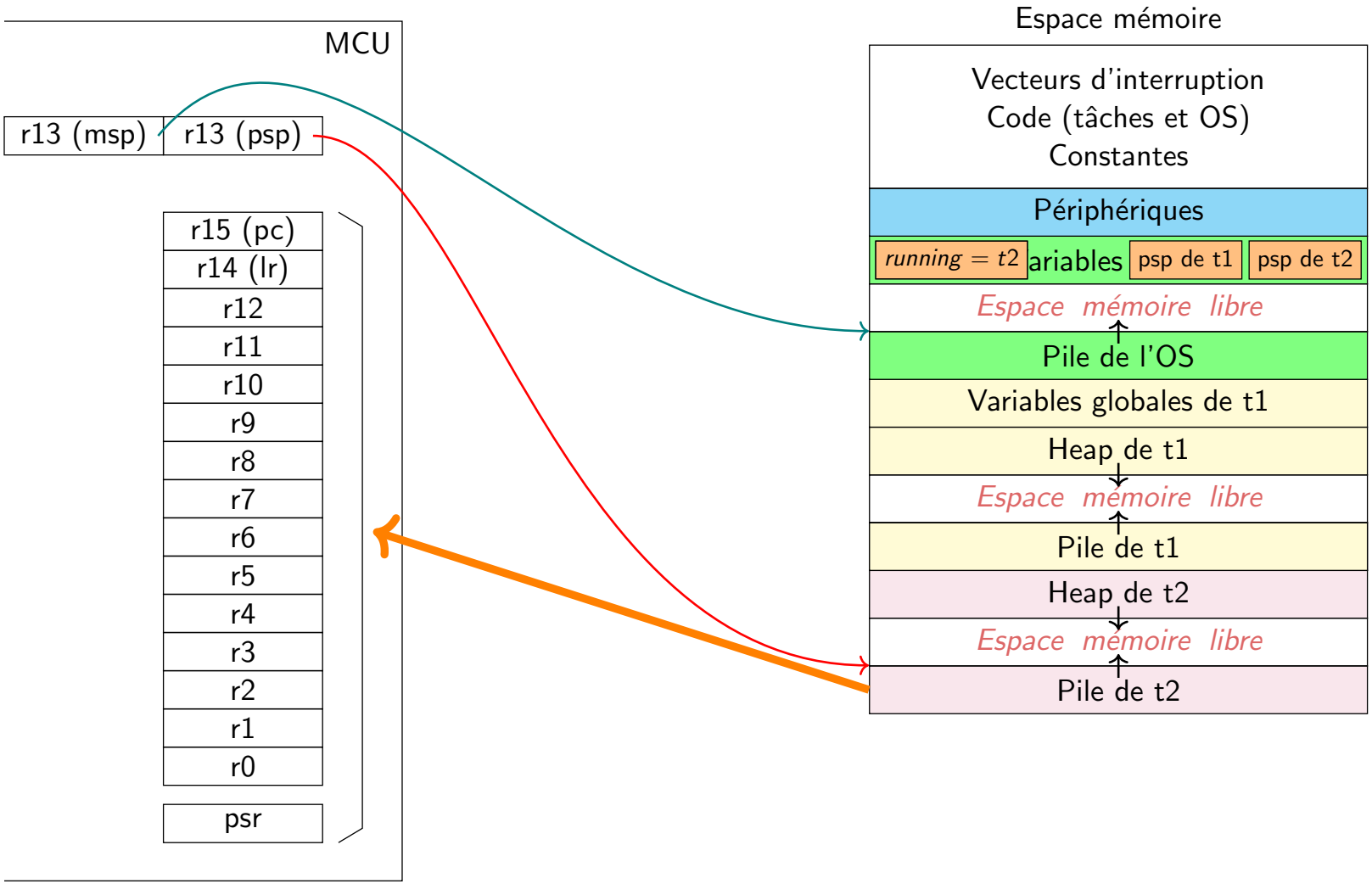
Changement de contexte, example du Cortex M (3)



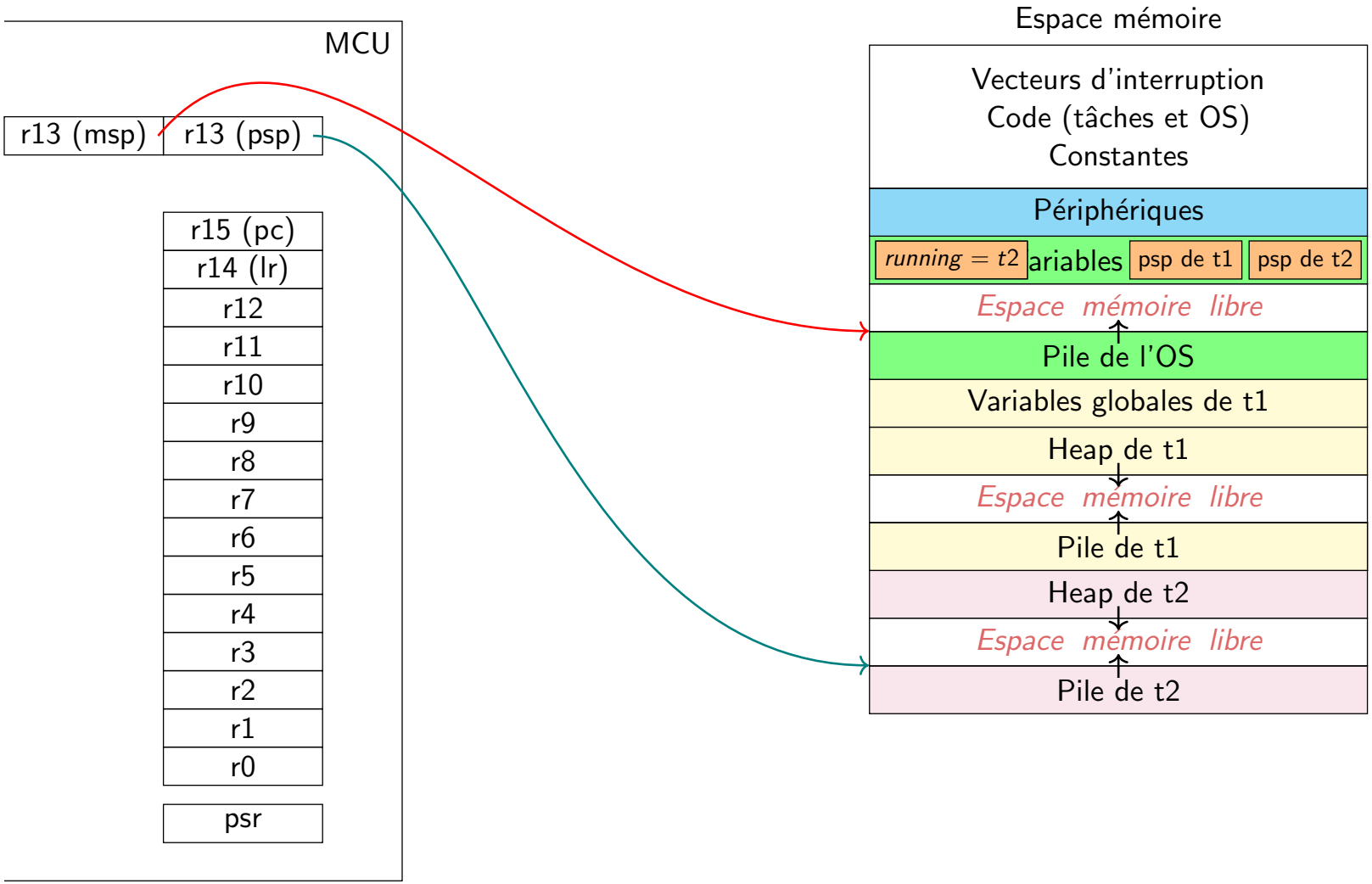
Changement de contexte, example du Cortex M (3)



Changement de contexte, example du Cortex M (3)



Changement de contexte, example du Cortex M (3)



Section 4

Conclusions

Travaux passés et en cours dans l'équipe

Thèse de Dominique Bertrand

- Dimensionnement des paramètres du module de protection temporelle

Thèse de Sylvain Cotard

- Développement d'un service de vérification en ligne
- Développement d'un service de communication *wait free*

Thèse de Toussaint Tigori

- Génération de noyau spécialisé pour l'application

Thèse de Louis-Marie Givel

- Test de propriétés temps réel par injection de délais (runtime enforcement)

Travaux récents et en cours dans l'équipe

Thèse de Dimitry Solet

- Développement d'un service de vérification en ligne par une approche matérielle (en utilisant un SoPC).

Thèse de Khaoula Boukir

- Mise en œuvre de politiques d'ordonnancement temps réel multiprocesseur prouvées.

Thèse d'Imane Haur

- Vérification et certification AUTOSAR multicœur

Thèse d'Antoine Bernabeu

- Trampoline pour les systèmes intermittents

Trampoline

- Exécutif temps réel pour « petit » micro-contrôleurs, conforme OSEK/VDX OS et AUTOSAR OS
- Système statique : ne pas hésiter à abuser de cette propriété
- Architecture modulaire, chaîne de développement extensible
- Logiciel libre sous licence GPL