UJF: Soutenance TER

Définition de portabilité en termes de modèle d'exécution pour la simulation des systèmes sur puces.

Par: Giani Velasquez
Encadré par: Matthieu Moy et Giovanni Funchal

PLAN

Introduction : Domaine du sujet.

État de l'art : Simulateur sur puce.

Mon sujet de Stage et Motivations.

Contributions

Conclusions et Améliorations possibles

Introduction

Aujourd'hui, les systèmes embarqués sont partout dans la vie quotidienne.

Systèmes sur puce (SoC)

Ces appareils doivent satisfaire des contraintes très fortes telles que:

- Performance du traitement informatique intensive.
- Consommation très basse d'énergie.
- Prix raisonnable.

Les microprocesseurs des ordinateurs traditionnels ne sont pas adaptés pour ces applications.

Systèmes sur puce

SoC: Matériel + logiciel.

Complexité de ces systèmes

+

Contraintes de temps de mise sur le marché.

travailler sur des modèles.

Flot de conception

Plusieurs représentations du système, pour différentes utilisations et différents niveaux de détails.

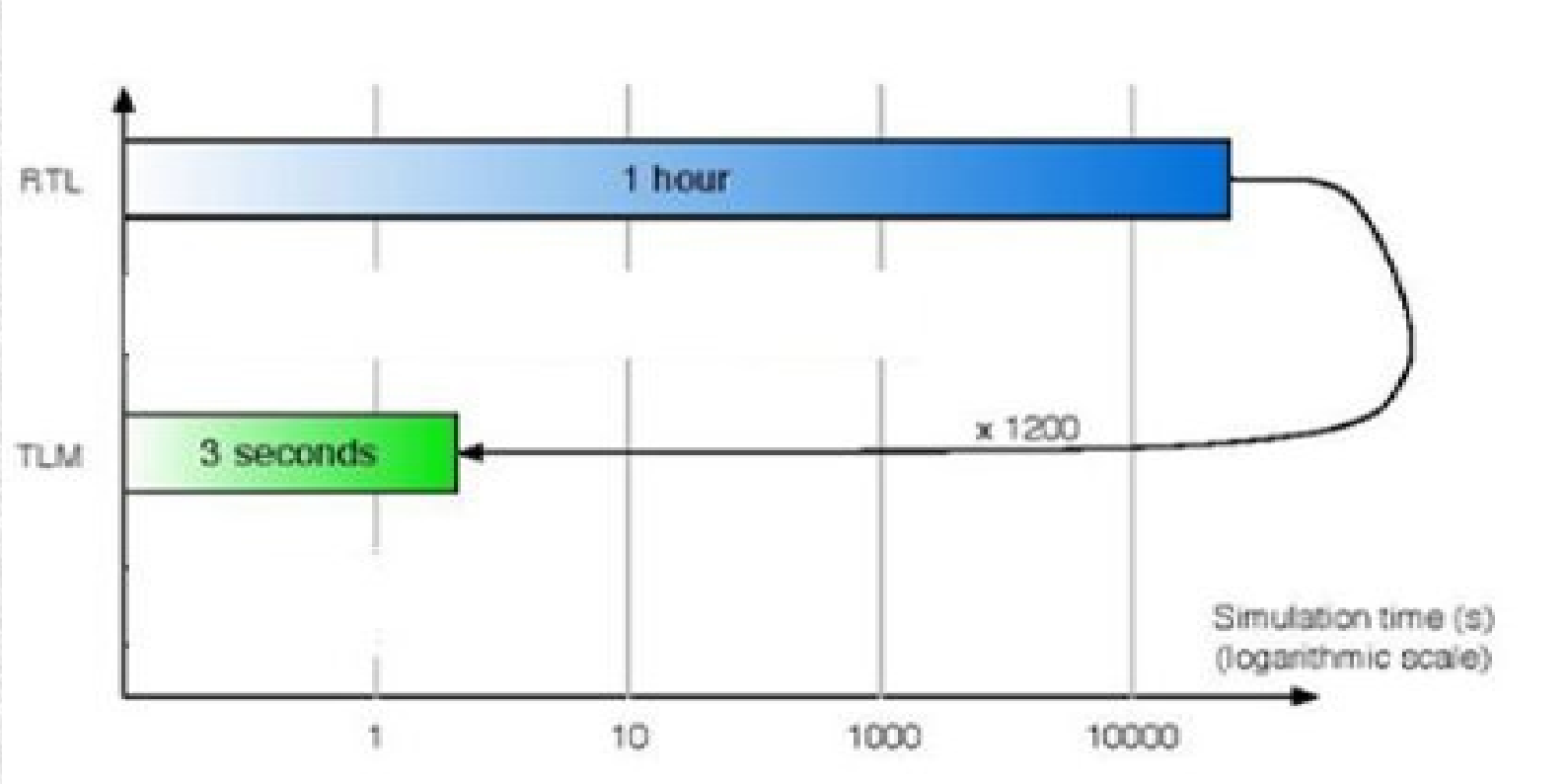
RTL (Register Transfer Level): Très détaillé.
⇒ développement du matériel.

TLM (Transaction Level Modeling): Plus abstrait.
⇒ développement du logiciel.

RTL est incontournable dans l'industrie, et TLM est émergent.

RTL vs TLM

Temps de simulation pour coder et décoder une image avec le codec MPEG4.



SystemC

Utilisé pour développer des modèles TLM.

Modèle d'exécution coopératif.

- Un seul processus s'exécute à la fois.
- Processus courant donne la main à un autre.

PLAN

Introduction

État de l'art : Simulateur sur puce.

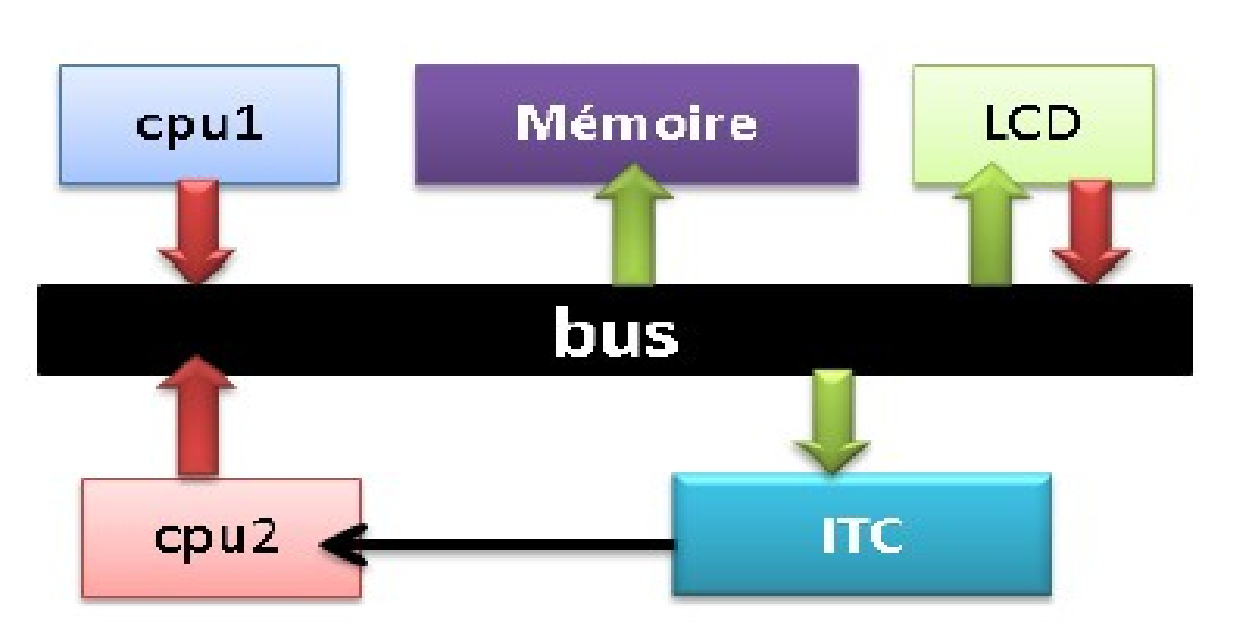
Mon sujet de Stage et Motivations

Contributions

Conclusions et Améliorations possibles

État de l'art: Jtlm

Simulateur pour système sur puces (Nabila Abdessaied).



Écrit en java et utilise les threads.

Description parallèle et exécution parallèle.

- → Chaque comportement est un thread.

Jtlm préemptif

Jtlm a un ordonnanceur préémptif.

Principe: Tous les comportements peuvent s'exécuter en même temps (sauf parties synchronisées).

Le temps en Jtlm

Temps de simulation: Temps que la vraie puce aurait pris pour effectuer son travail.

Wall clock time: Temps de la simulation de la plate-forme virtuelle.

- Ex. Deux ordinateurs, dont un est deux fois plus puissant que l'autre.

Les tâches en Jtlm

Tâches instantanées.

Exemple: Un comportement effectue une tâche qui ne prend pas de temps: **f();**

Tâches avec durée.

Exemple: Un comportement effectue une tâche qui va avoir une durée de 10 unités:

```
consume(){
    f();
}.during(10);
```

Tâches avec durée indéterminée(Stage réalisé en parallèle par Mohamed EL AISSAOUI).

Les primitives pour l'utilisateur: Wait et Wake

waitTime(time): Attente sur du temps.

waitEvent(): Attente d'un événement.

waitInterruption(): Attente d'une interruption.

e.WakeUpFromWaitingEvent(): Réveille les comportements qui sont en attente de l'événement e.

sendInterruption(): Réveille le comportement qui est en attente d'une interruption.

Les primitives de l'ordonnanceur: Wake

WakeUpFromWaitingTime(): Réveille les comportements qui sont en attente sur du temps.

PLAN

Introduction

État de l'art

Mon sujet de Stage et Motivations

Contributions

Conclusions et Améliorations possibles

Mon sujet de stage

1) Version coopérative du simulateur Jtlm.

2) Isoler et définir la notion de portabilité du simulateur en termes de modèle d'exécution.

Motivations

Au début: Ordonnanceur de Jtlm préemptif.

Implémenter un ordonnanceur de Jtlm **coopératif**

⟹ Expérimenter ces deux modèles.

⟹ Comparer ces deux modèles d'exécution.

PLAN

Introduction

État de l'art

Mon sujet de Stage et Motivations

Contributions

Conclusions et Améliorations possibles

Contributions

Ordonnanceur coopératif

Principe: Un seul comportement s'exécute à la fois, et c'est le comportement courant qui rend la main à un autre comportement pour qu'il s 'exécute.

Voici trois composants : P1, P2, LCD.

P1 ⟹ B1,

P2 ⟹ B2,

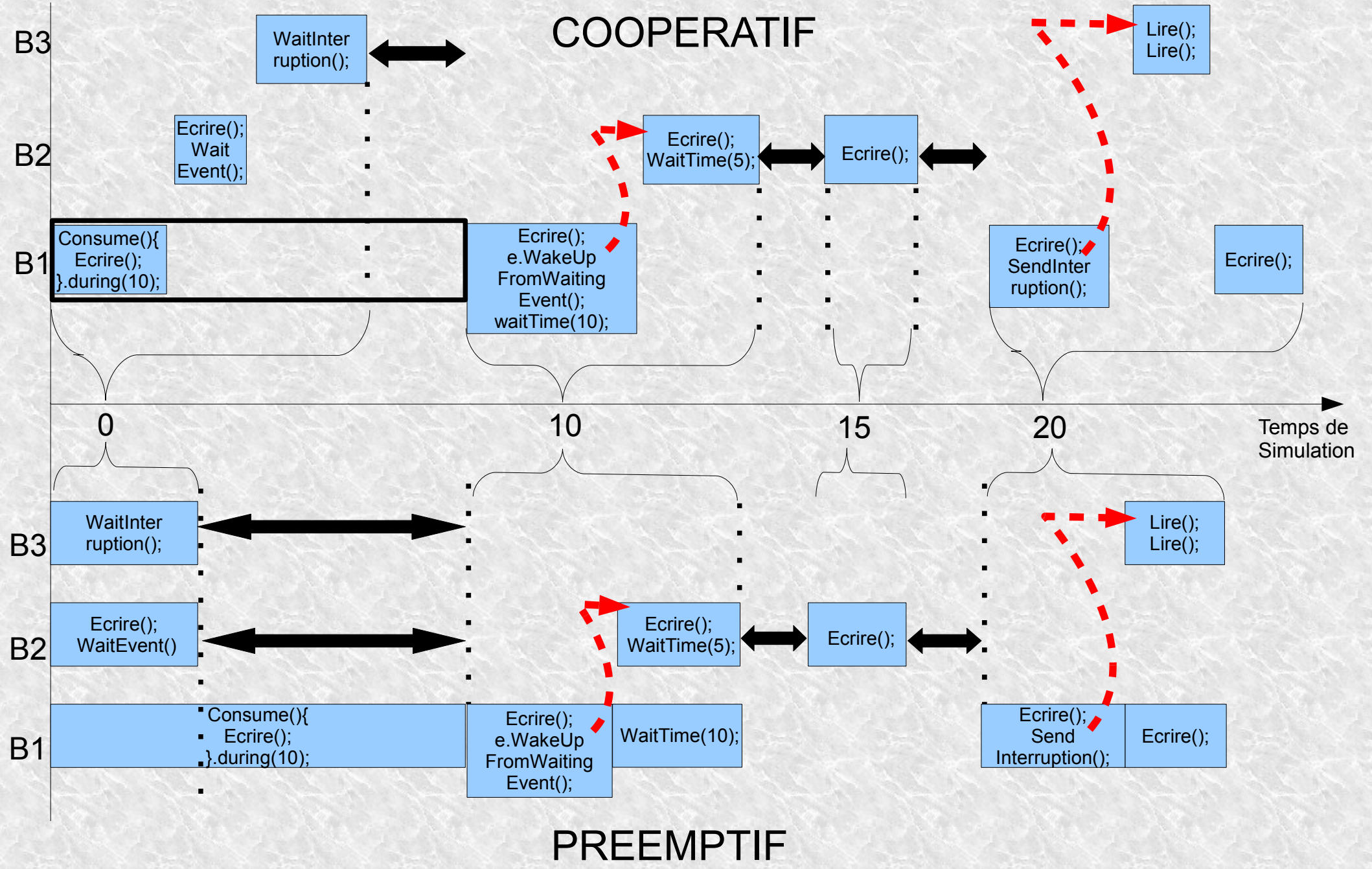
LCD ⟹ B3.

Trois comportements

```
B1{
        consume(){
                Ecrire();
        }.during(10};

        Ecrire();

        wakeUpFromWaitingEvent();
        waitTime(10);
        Ecrire();
        sendInterruption();
        Ecrire();
}
```

```
B2{
        Ecrire();
        waitEvent();
        Ecrire();
        waitTime(5);
        Ecrire();
}

B3{
        waitInterruption();
        Lire();
        Lire();
}
```

Simulation dans les deux modèles



Début de la simulation

Liste Eligible : Comportements qui sont prêt à s'exécuter.

➡ B1, B2, B3.

Liste d'Attente: Comportements qui sont en attente sur du temps, ou qui ont effectué une tâche avec durée.

➡ Vide.

```
B1{
        consume(){
            Ecrire();
        }.during(10};

         ....
}
```

B1

Consume(){
Ecrire();
}.during(10)

t=0

0 10 15 20 Temps de Simulation

Liste Eligible : B2, B3

Liste d'Attente:

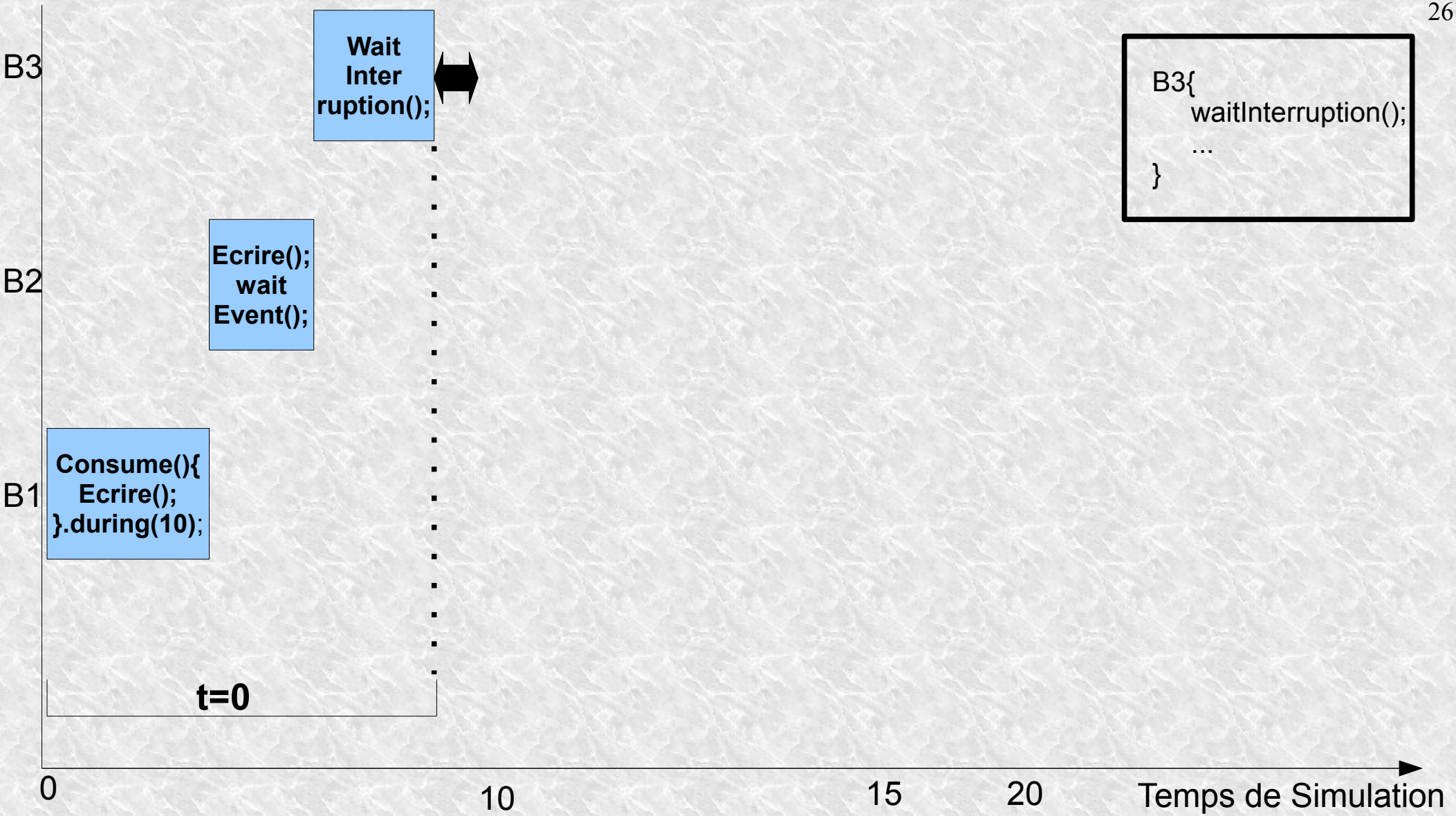
B1

0 10 15 20

```
B2{
      Ecrire();
      waitEvent();
      ...
}
```

Liste Eligible : B3

Liste d'Attente:



Liste Eligible : Vide

Liste d'Attente:

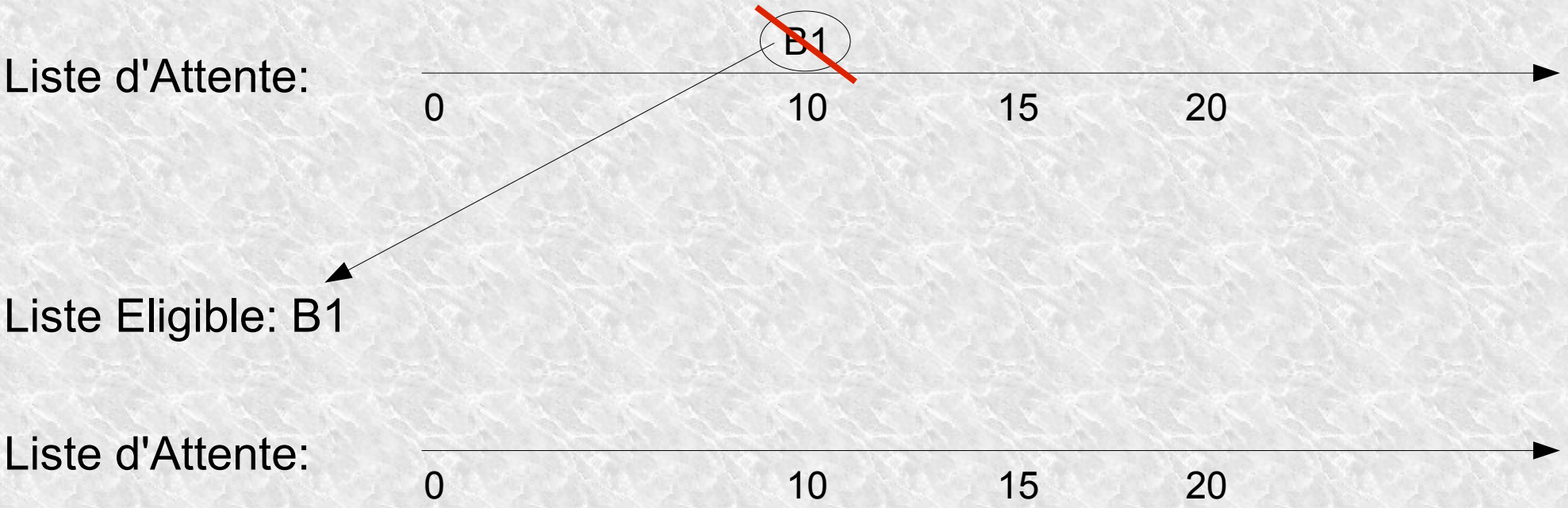
Liste Eligible vide!

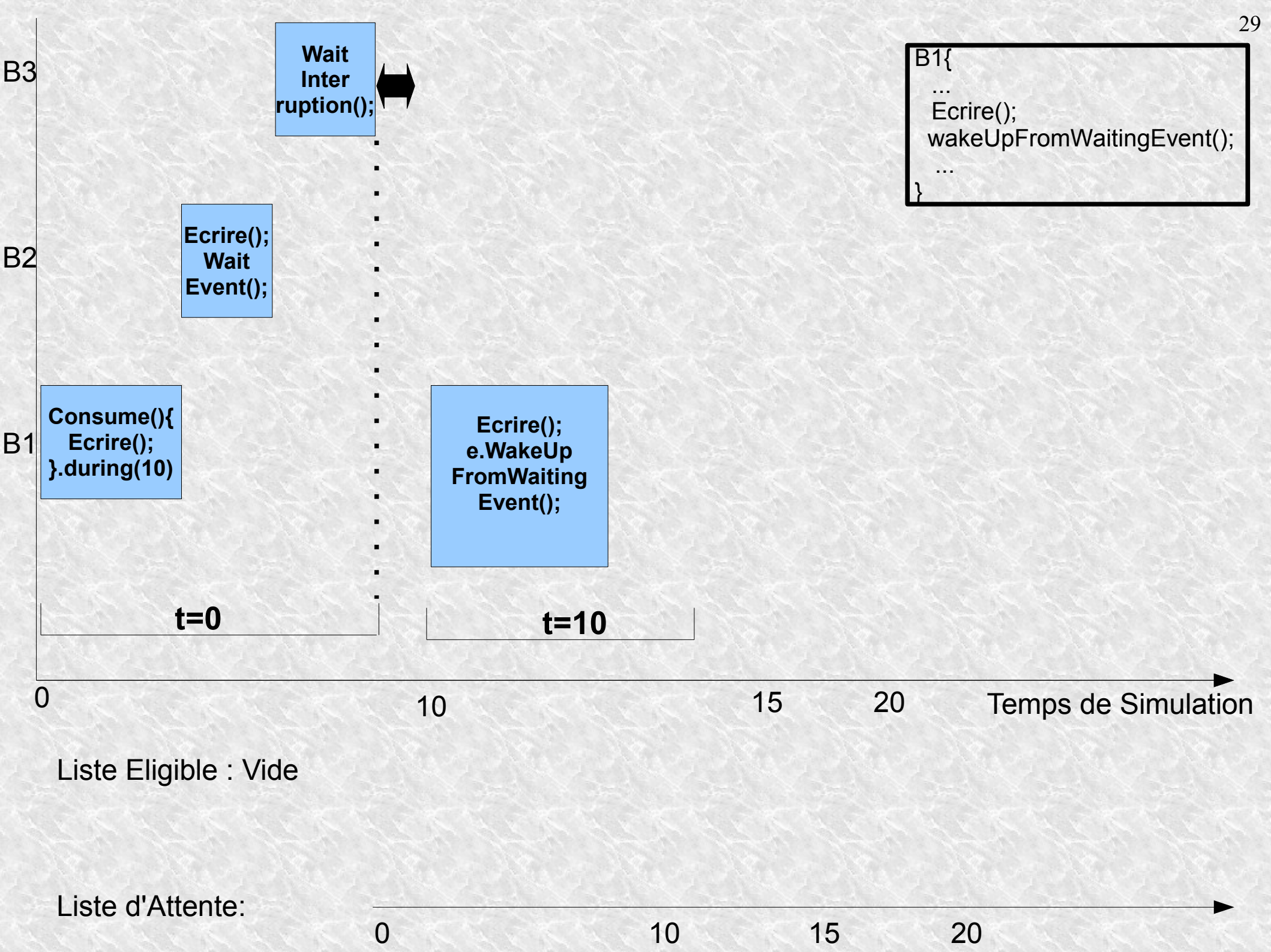
Liste d'Attente:

Liste Eligible: B1

Temps de Simulation = 10

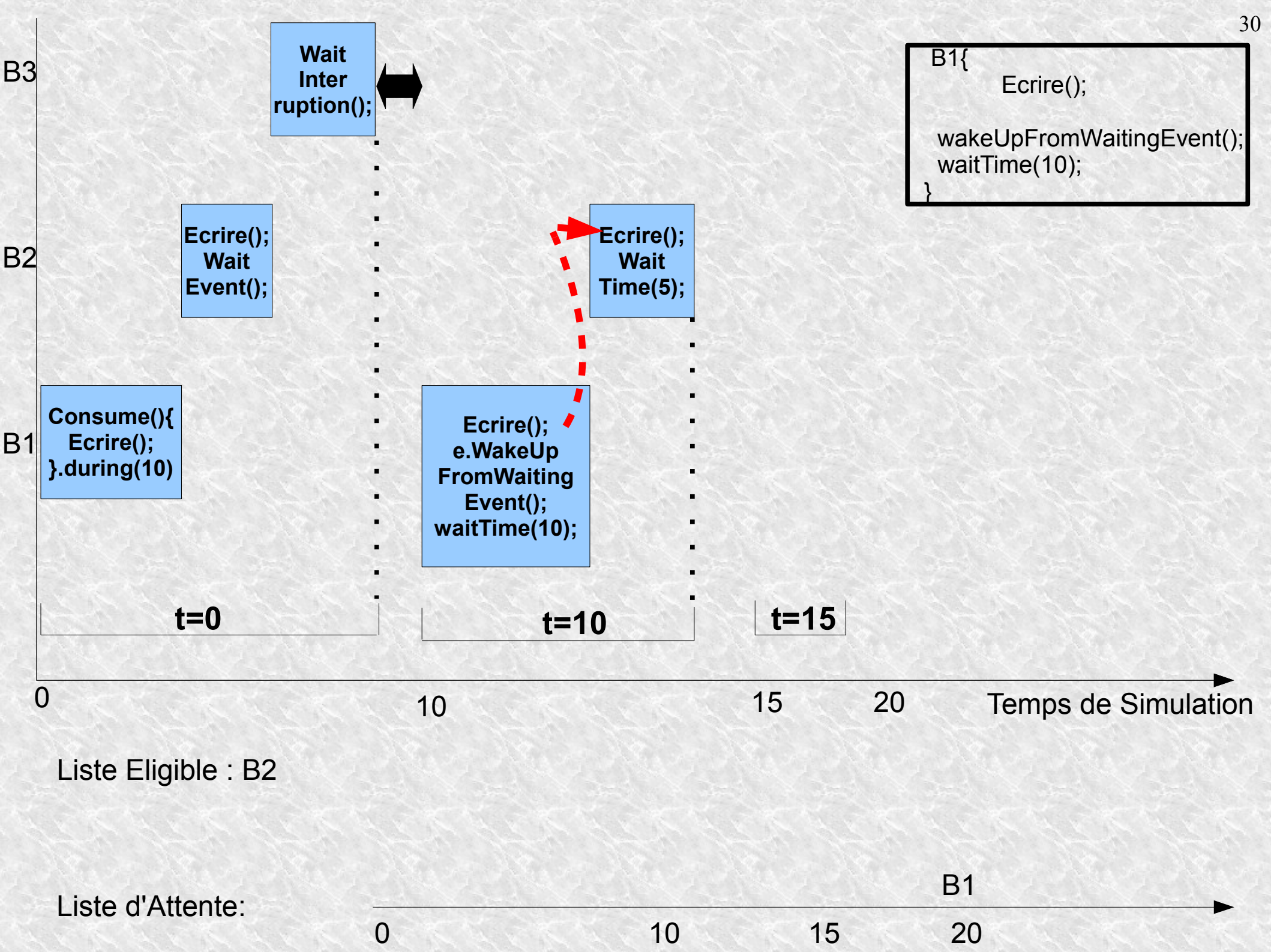
Liste Eligible vide!





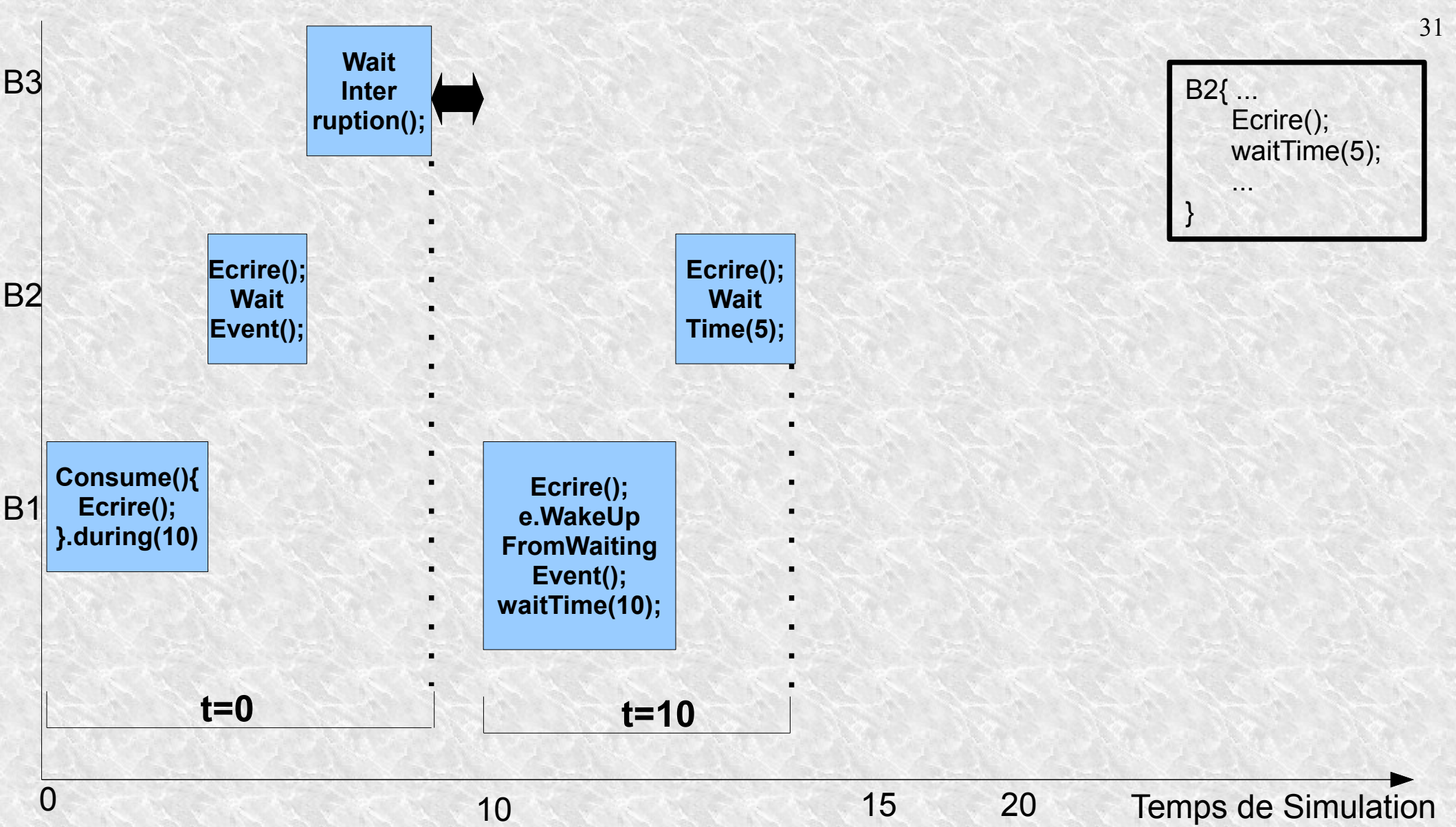
Liste Eligible : Vide

Liste d'Attente:



Liste Eligible : B2

Liste d'Attente:

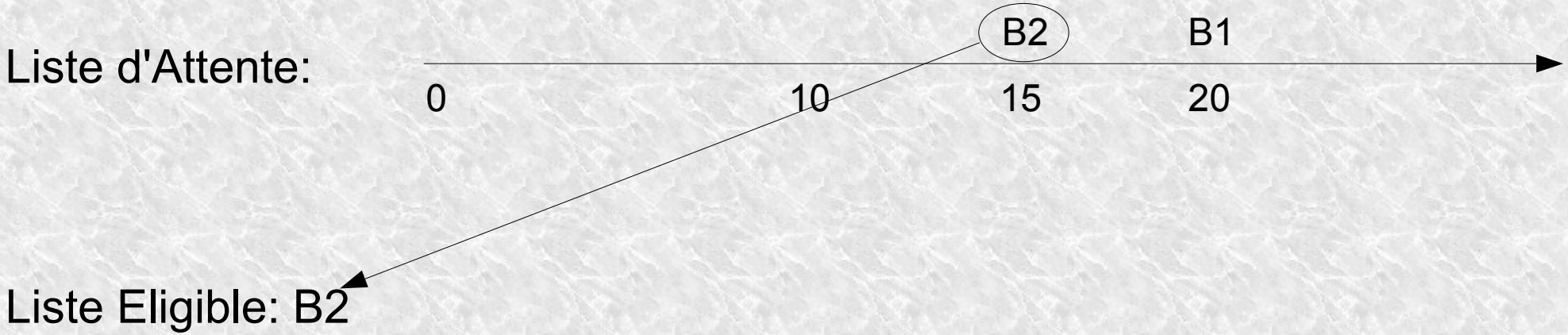



```
B2{ ...
      Ecrire();
      waitTime(5);
      ...
}
```

Liste Eligible : Vide

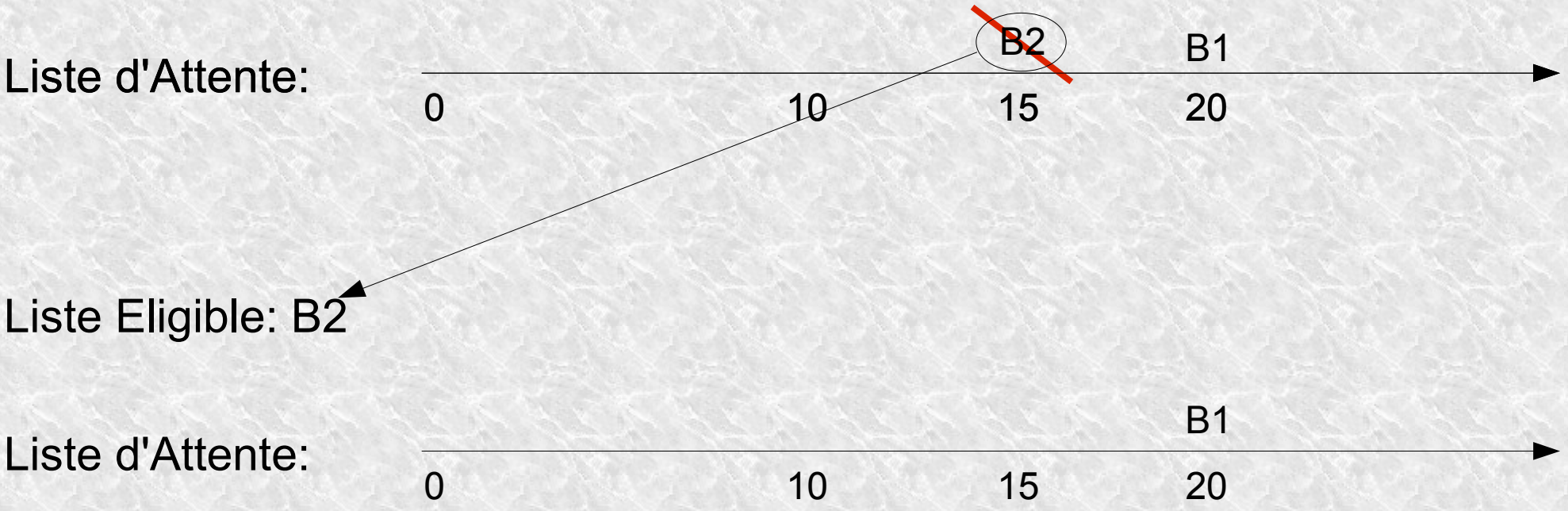
Liste d'Attente:

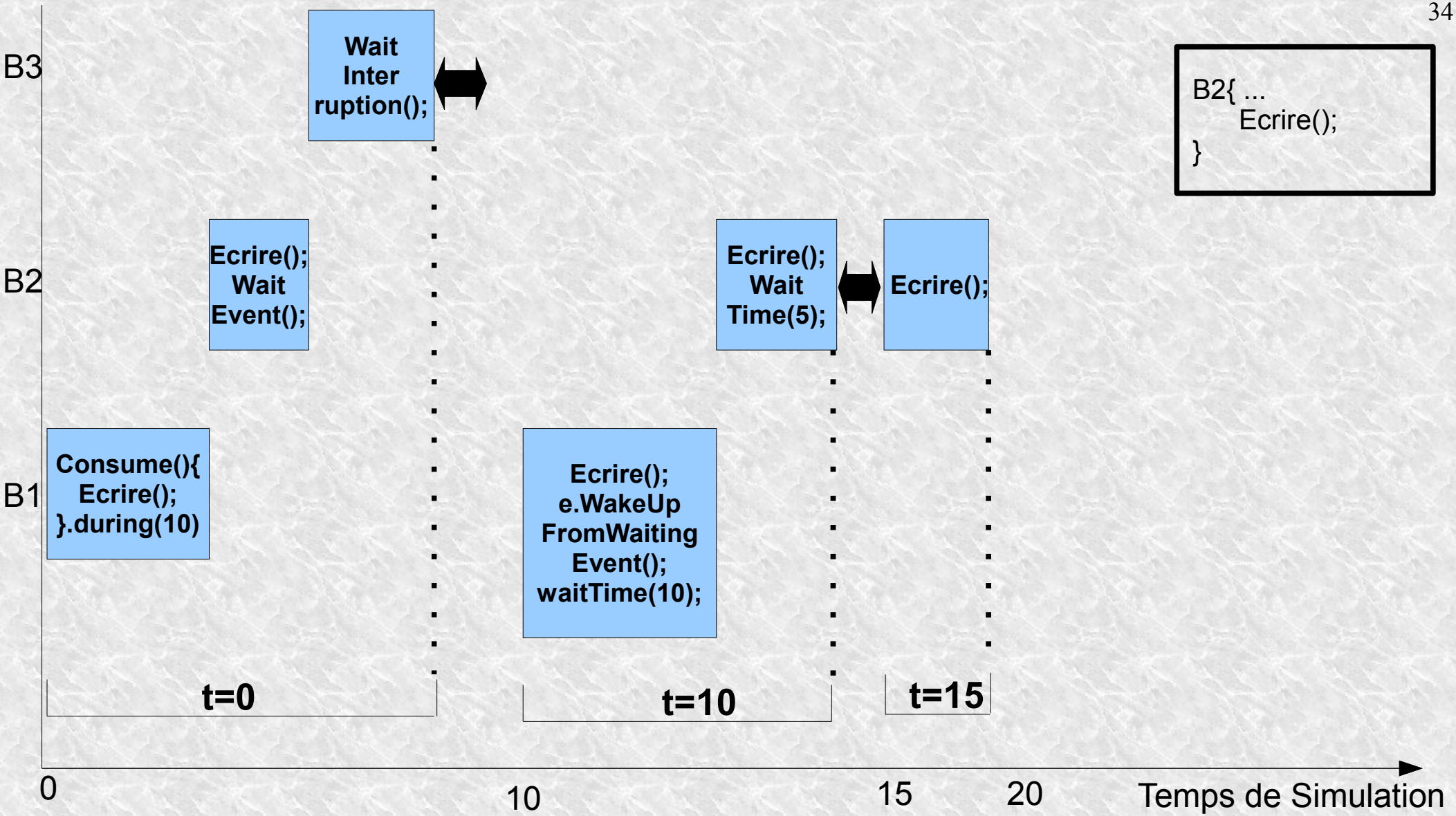
Liste Eligible vide!



Temps de Simulation = 15

Liste Eligible vide!



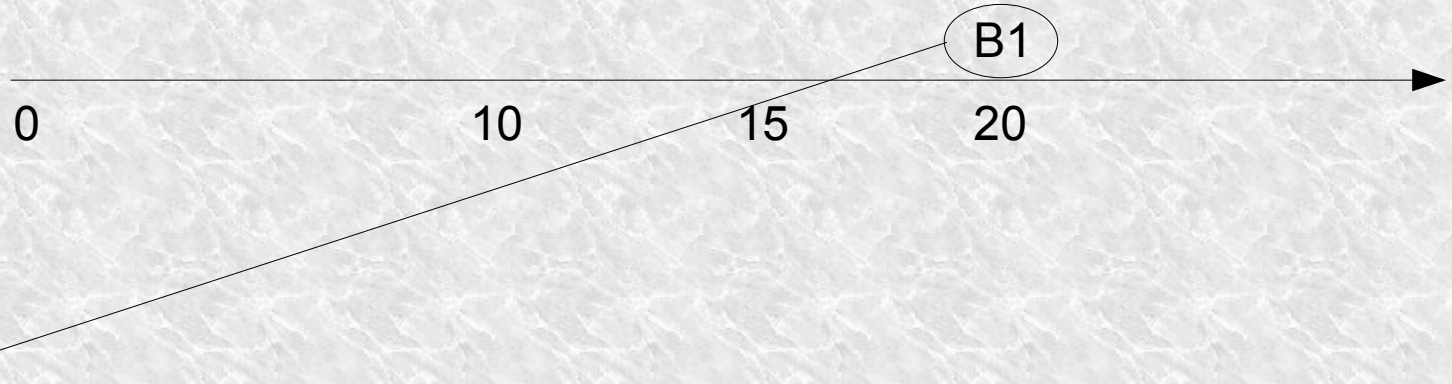


Liste Eligible : Vide

Liste d'Attente:

Liste Eligible vide!

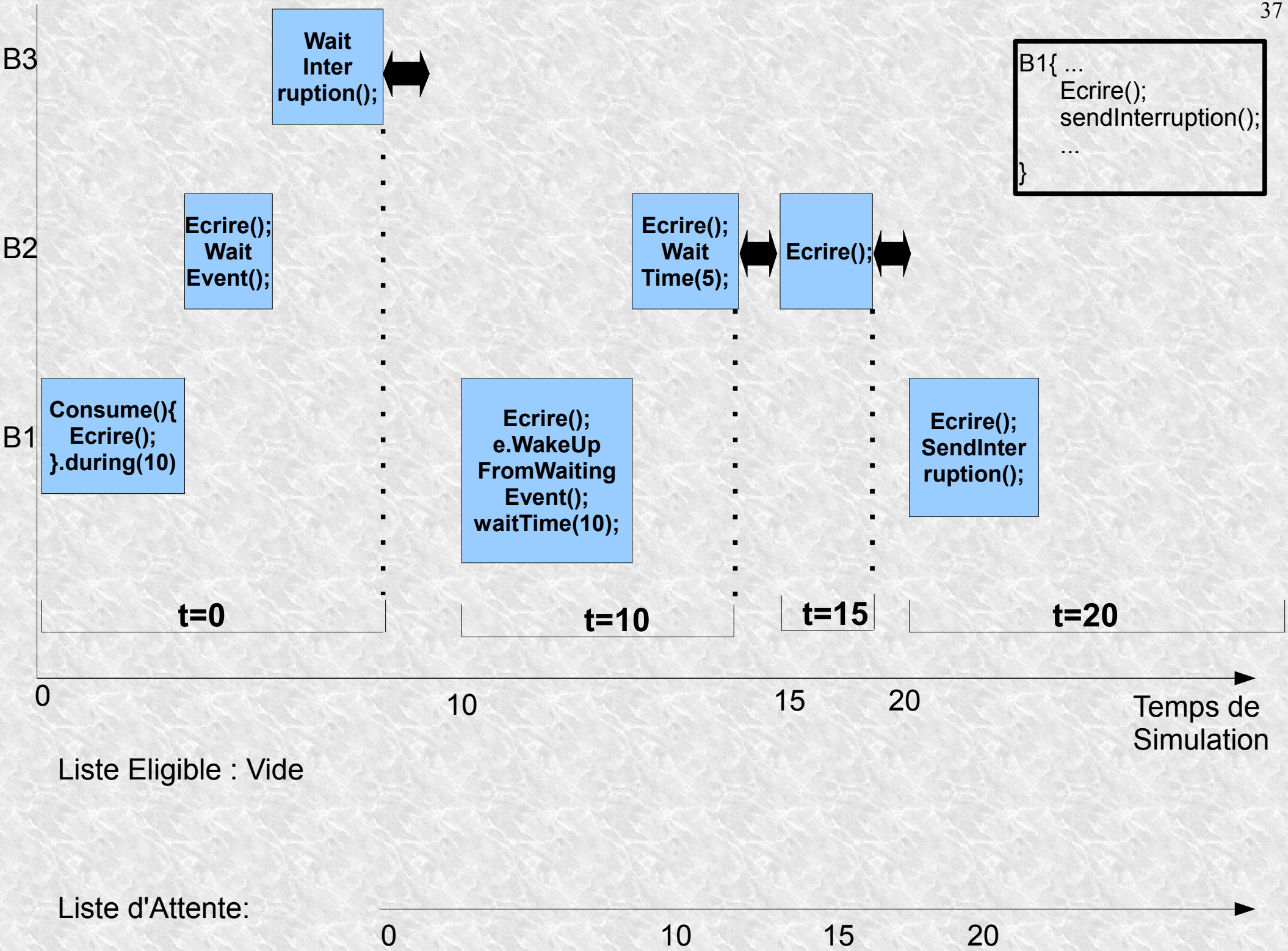
Liste d'Attente:

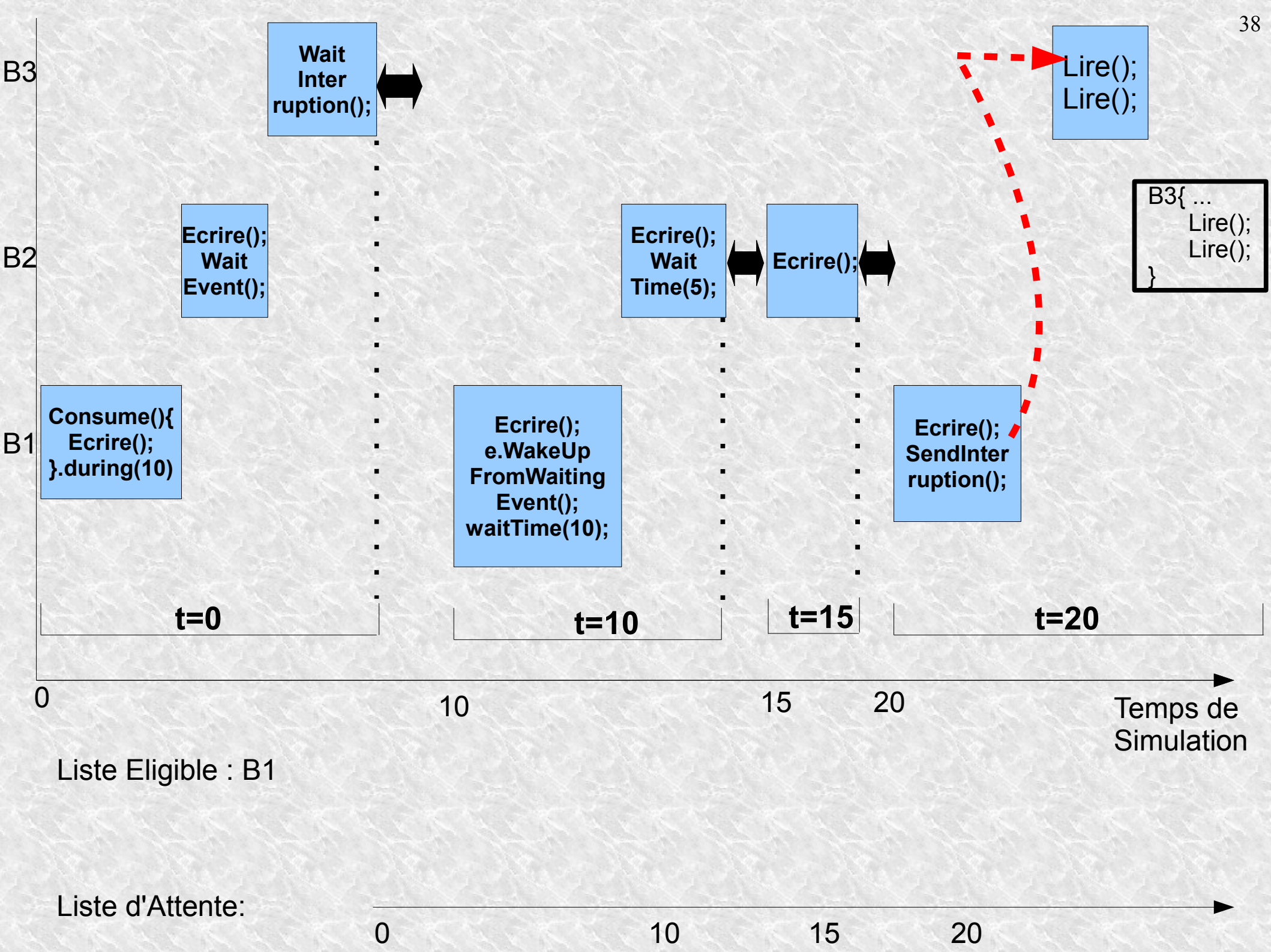


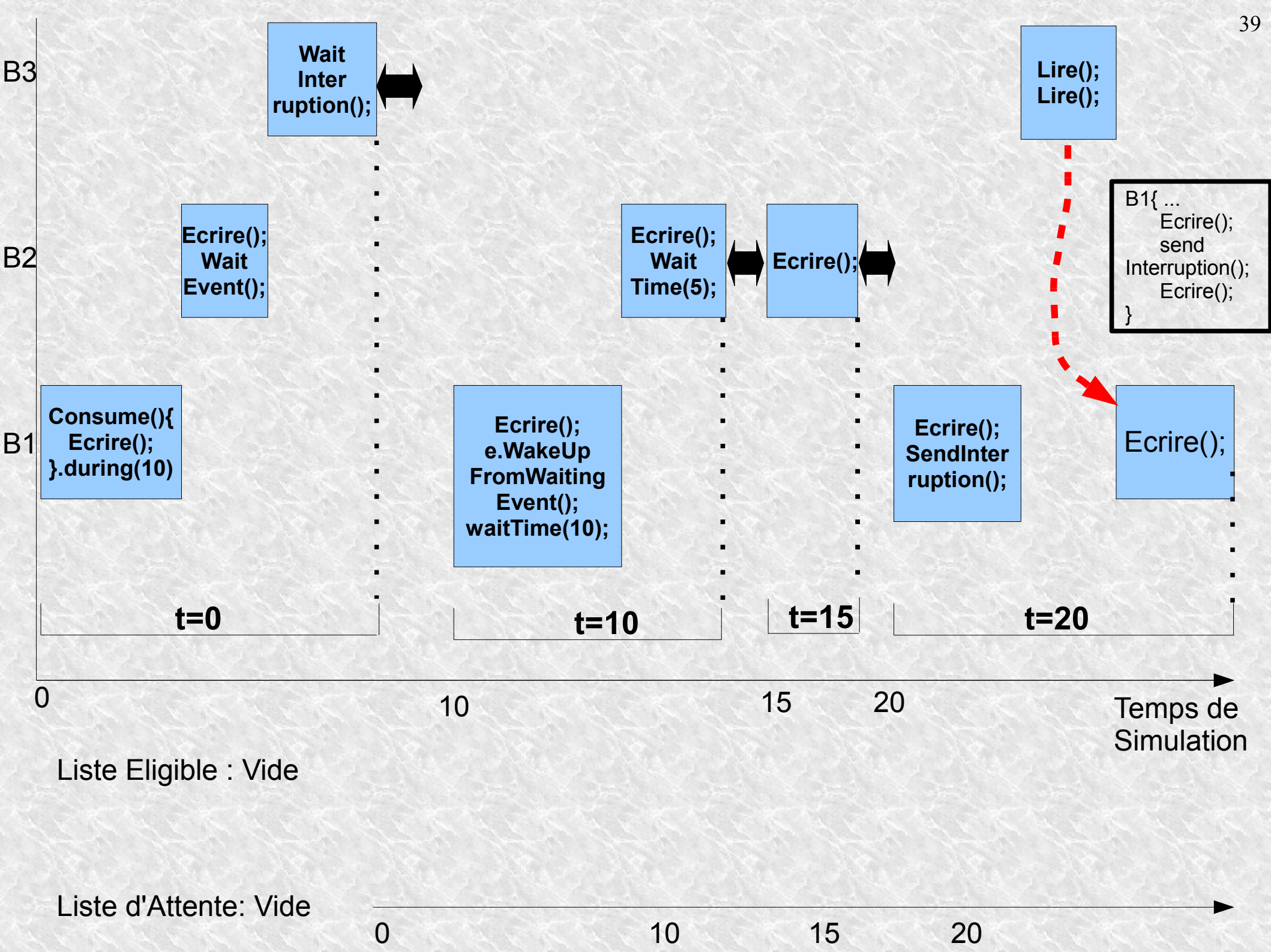
Liste Eligible: B1

Temps de Simulation = 20

Liste Eligible vide!


B3
B2
B1
Wait Interruption();
Ecrire(); Wait Event();
Consume(){ Ecrire(); }.during(10)
Ecrire(); Wait Time(5);
Ecrire();
Ecrire(); e.WakeUp FromWaiting Event(); waitTime(10);
Ecrire(); SendInterruption();
B1{ ... Ecrire(); sendInterruption(); ... }
t=0
t=10
t=15
t=20
0
10
15
20
Temps de Simulation
Liste Eligible : Vide
Liste d'Attente:
0
10
15
20


B3
B2
B1
Wait Inter ruption();
Ecrire(); Wait Event();
Consume(){ Ecrire(); }.during(10)
Ecrire(); e.WakeUp FromWaiting Event(); waitTime(10);
Ecrire(); Wait Time(5);
Ecrire();
Ecrire(); SendInter ruption();
Lire(); Lire();
B3{ ... Lire(); Lire(); }
t=0
t=10
t=15
t=20
0
10
15
20
Temps de Simulation
Liste Eligible : B1
Liste d'Attente:
0
10
15
20



Liste Eligible : Vide

Liste d'Attente: Vide

Portabilité

Isoler et donner des règles pour définir la notion de portabilité.

Même code utilisateur dans la vraie puce et simulateur.

Tirer des bouts de code d'un mode à l'autre.

Pas assez de temps pour aller plus loin.

PLAN

Introduction

État de l'art

Mon sujet de Stage et Motivations

Contributions

Conclusions et Améliorations possibles

Conclusions

Un modèle d'exécution coopératif et préemptif.

Contraintes au niveau de l'implantation.

Modèle coopératif: Grande granularité dans les transactions.

Cache des bugs ⇒ Mauvais fonctionnement du logiciel embarqué.

Le modèle préemptif, ne cache pas ces bugs.

Mécanismes de synchronisation ⇒ cacher de bugs.

Améliorations

Que faire avec les tâches avec durée inconnue?

Réfléchir au cas de polling dans la version coopérative.

Donner d'autres règles pour définir d'une façon plus exhaustive la portabilité en termes de modèle d'exécution.

Polling dans les deux modèles d'exécution.