SÉANCE 15

MONITEURS

SUJETS

Introduction au moniteurs

Comparaison entre les moniteurs et sémaphores

Variables de condition

- Les opérations des variables de condition

Exemples des moniteurs

MONITEUR

Un moniteur est un ensemble de routines qui sont protégés par un verrou d'exclusion mutuelle

- Aucune routine dans le moniteur peut s'exécuter par un thread jusqu'à ce que ce thread obtient le verrou

Tout autre thread doit attendre le thread qui exécute couramment afin d'abandonner le contrôle du verrou

Lorsqu'un moniteur est occupé, le code de synchronisation de compétition est ajouté par le compilateur

- Pourquoi est-ce un avantage?

MONITEUR

Un thread peut se suspendre à l'intérieur d'un moniteur et ensuite attendre qu'un événement se passe

- Si ceci se passe, alors un autre thread a l'opportunité de rentrer dans le moniteur

Habituellement, un thread se suspend en attendant une condition

- Durant l'attente, le thread abandonne temporairement son accès exclusif
- Il doit le réacquérir après que la condition soit remplie

MONITEUR VS SÉMAPHORE

uOttawa
L'Université canadienne
Canada's university

Un sémaphore est une construction plus simple que le moniteur parce qu'il s'agit seulement d'un verrou qui protège une ressource partagée

- Pas un ensemble de routines comme le moniteur

Une tâche doit acquérir (ou attendre pour) un sémaphore avant d'accéder une ressource partagée

Une tâche doit simplement faire appel à une routine (ou procédure) dans le moniteur afin d'accéder une ressource partagée

- Lorsque terminé, vous n'avez rien à relâcher
- Rappelez-vous que vous devez relâcher les sémaphores (si vous oubliez ➔inter-blocage)

SYNCHRONISATION DE COMPÉTITION

uOttawa
L'Université canadienne
Canada's university

Une des caractéristiques les plus importantes des moniteurs est que l'information partagée réside dans le moniteur

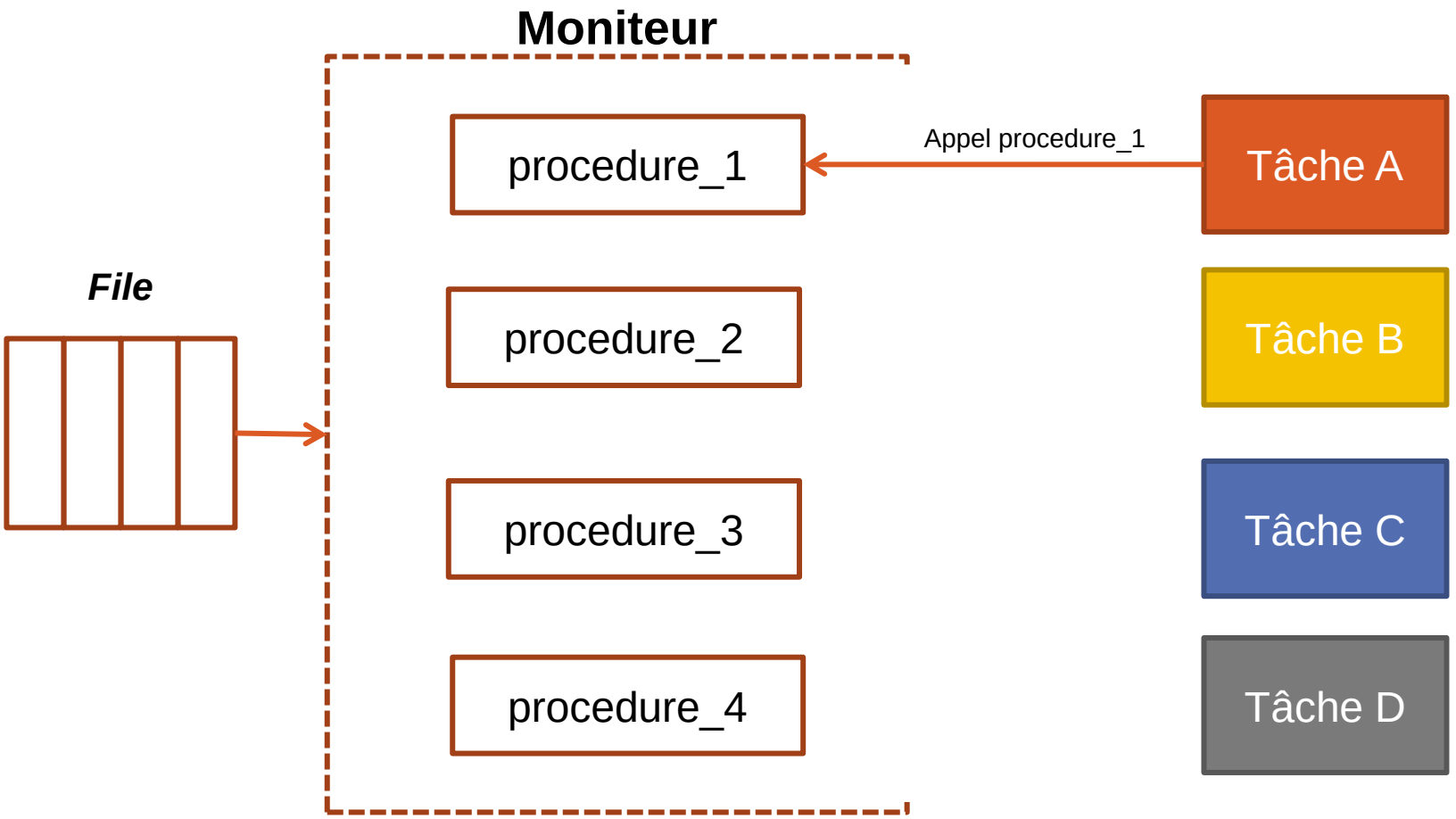
- Tout le code de synchronisation est centralisé dans un seul endroit

Le moniteur garantit la synchronisation en permettant l'accès à une seule tâche à la fois

- Rappelez-vous que par l'utilisation des sémaphores comptants, nous sommes capables de permettre l'accès à plusieurs tâches, et non nécessairement une seule

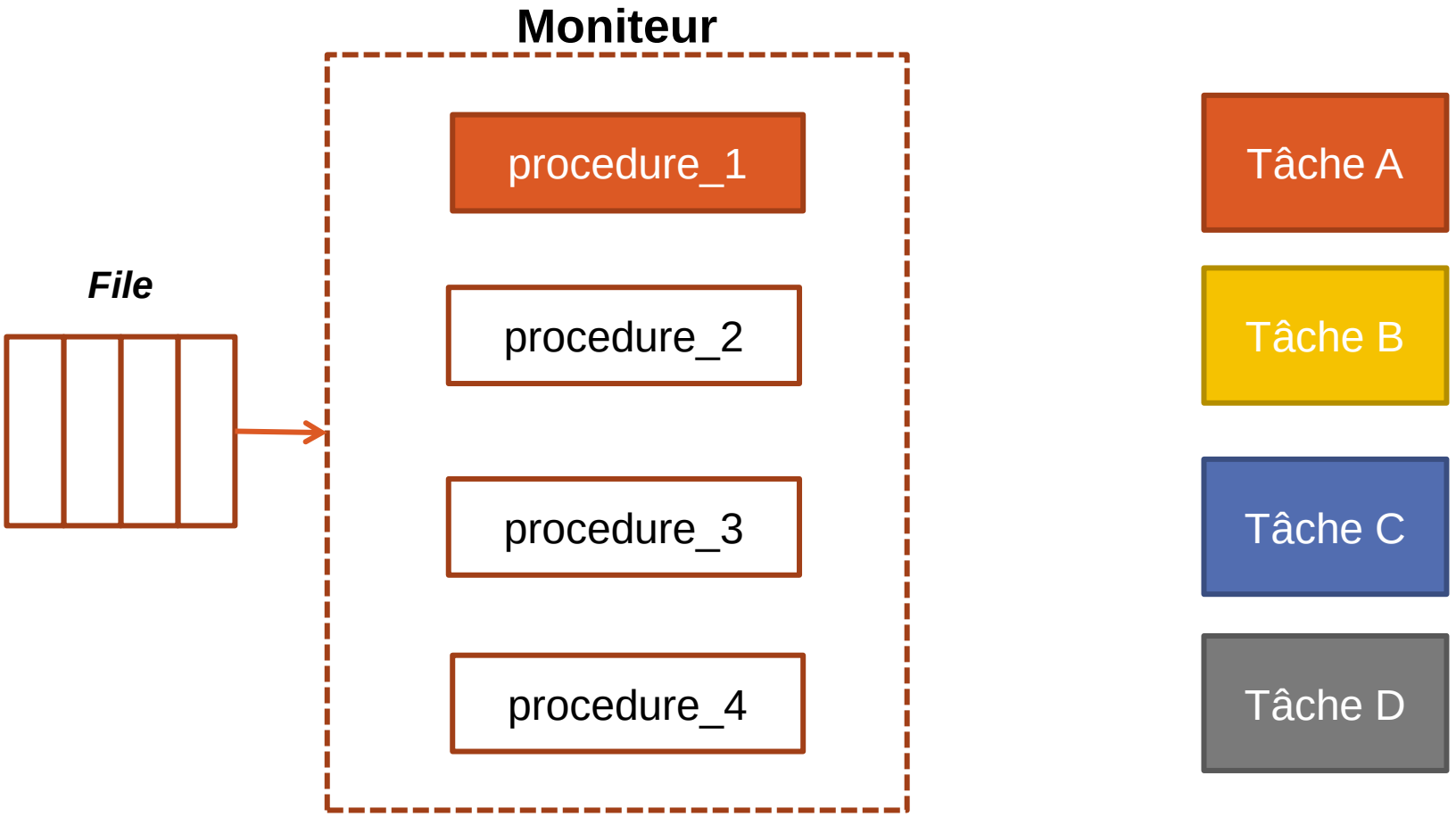
Les appels aux procédures du moniteur sont implicitement filés si le moniteur est occupé au moment de l'appel

EXEMPLE D'UTILISATION D'UN MONITEUR



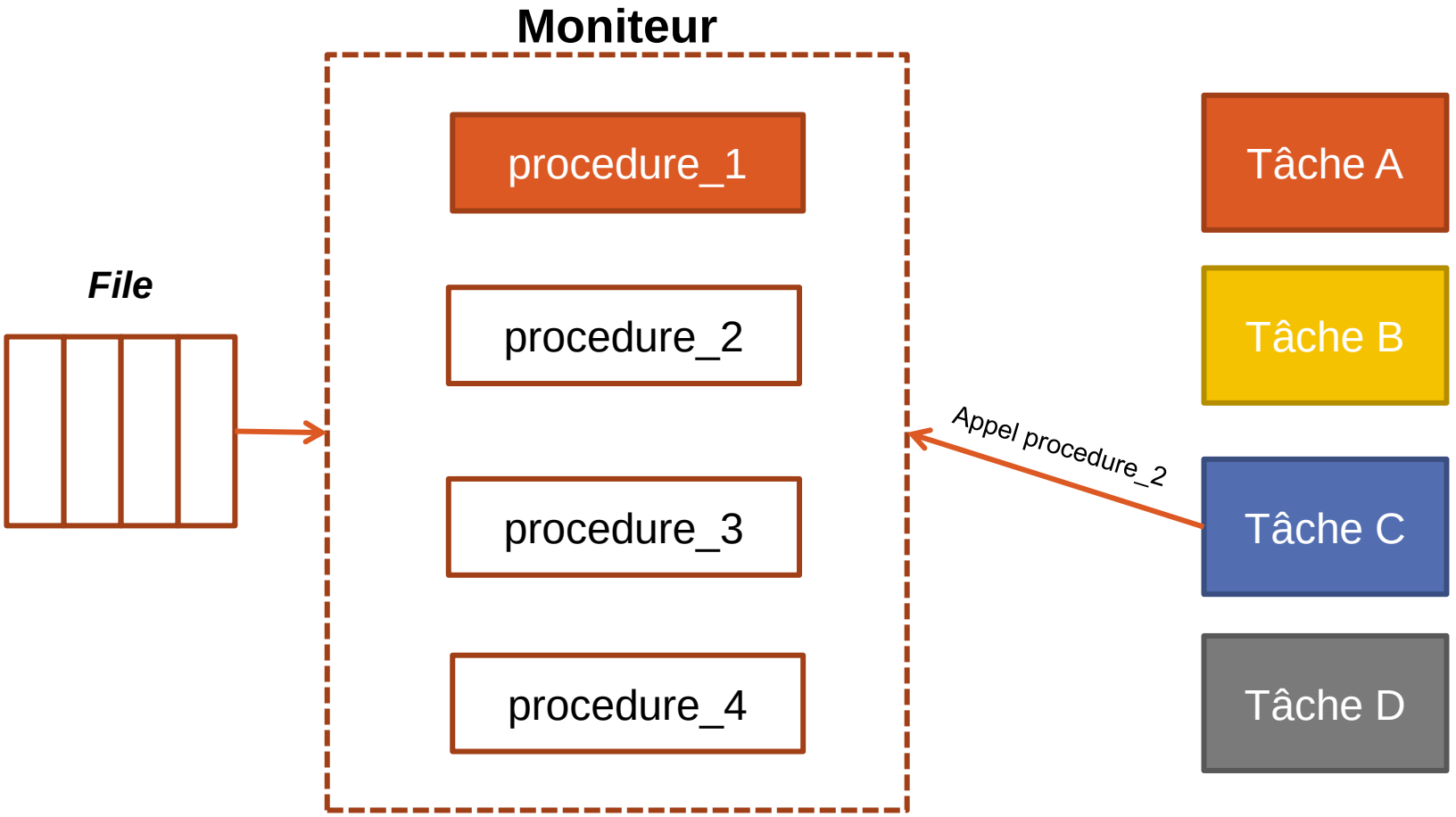
Propriétaire du Moniteur: Aucun

EXEMPLE D'UTILISATION D'UN MONITEUR



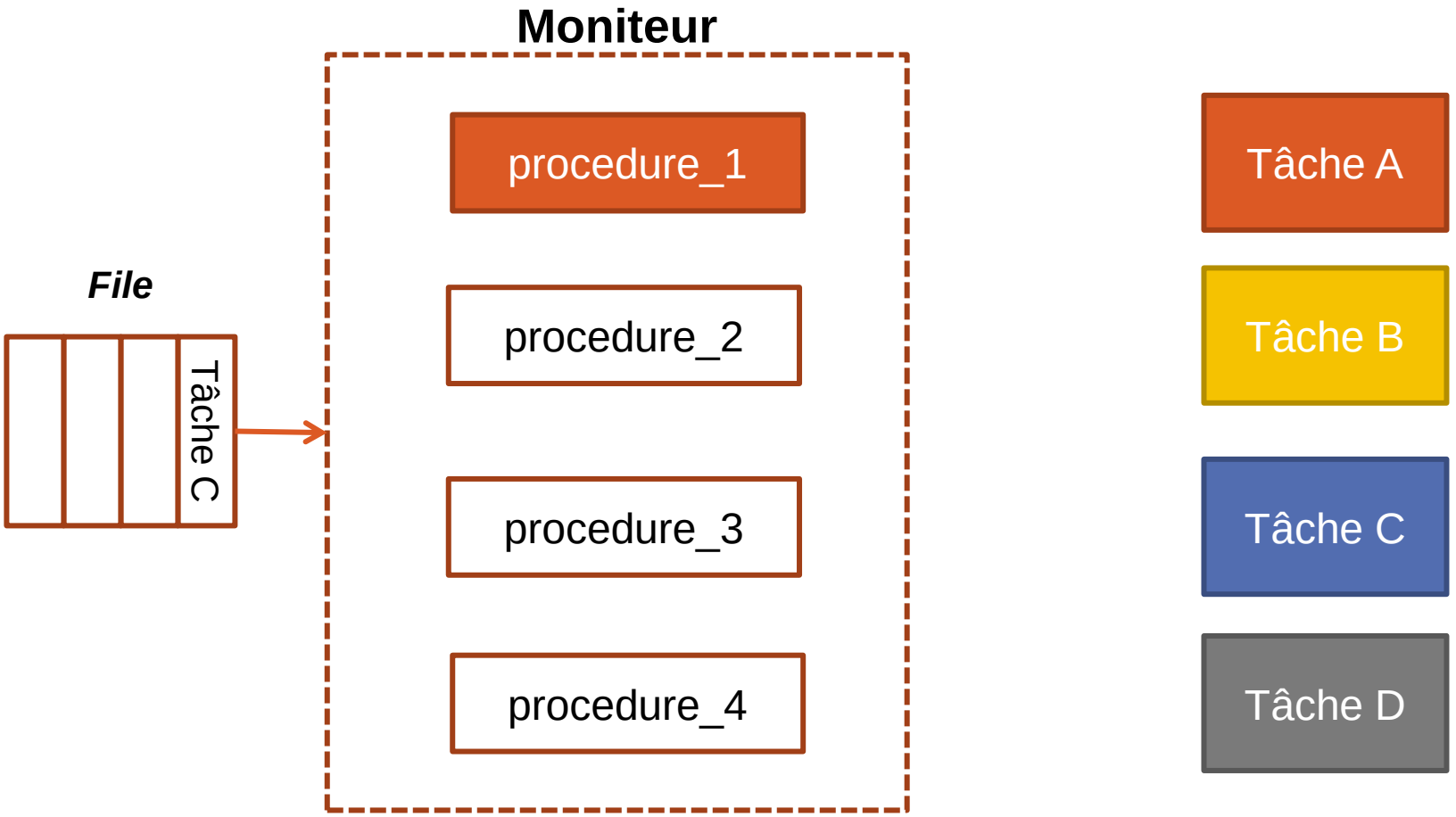
Propriétaire du Moniteur: Tâche A

EXEMPLE D'UTILISATION D'UN MONITEUR



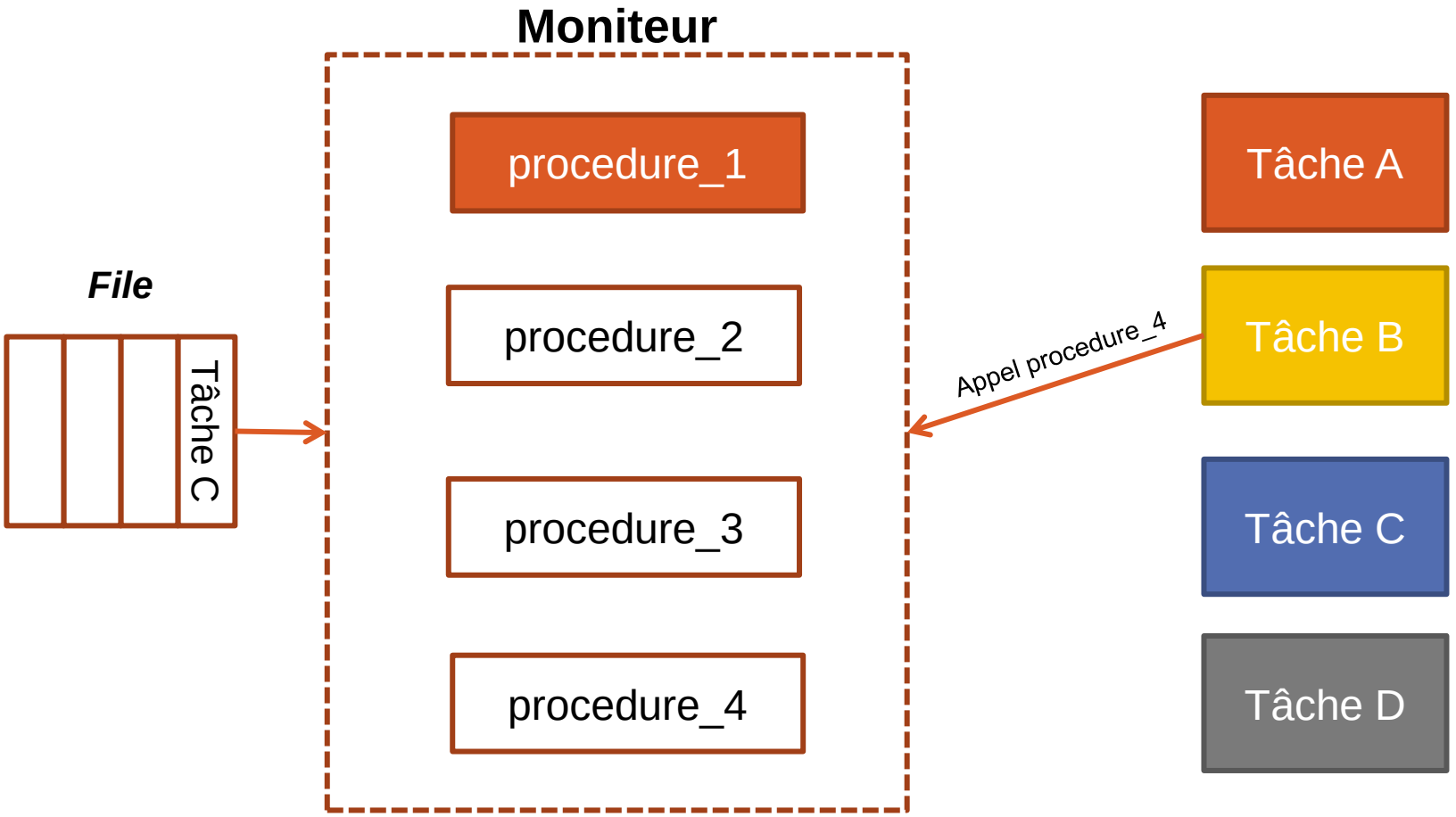
Propriétaire du Moniteur: Tâche A

EXEMPLE D'UTILISATION D'UN MONITEUR



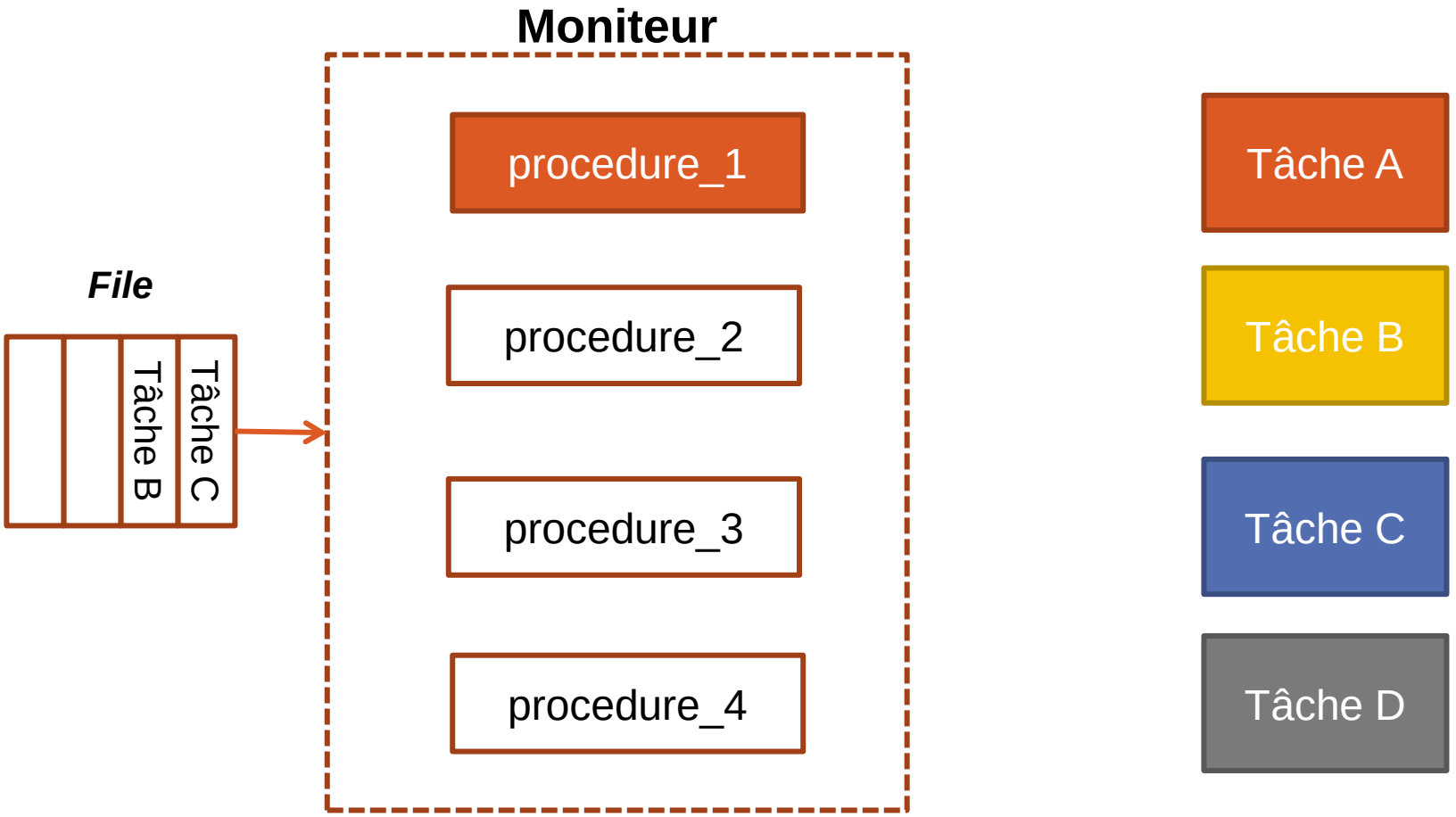
Propriétaire du Moniteur: Tâche A

EXEMPLE D'UTILISATION D'UN MONITEUR



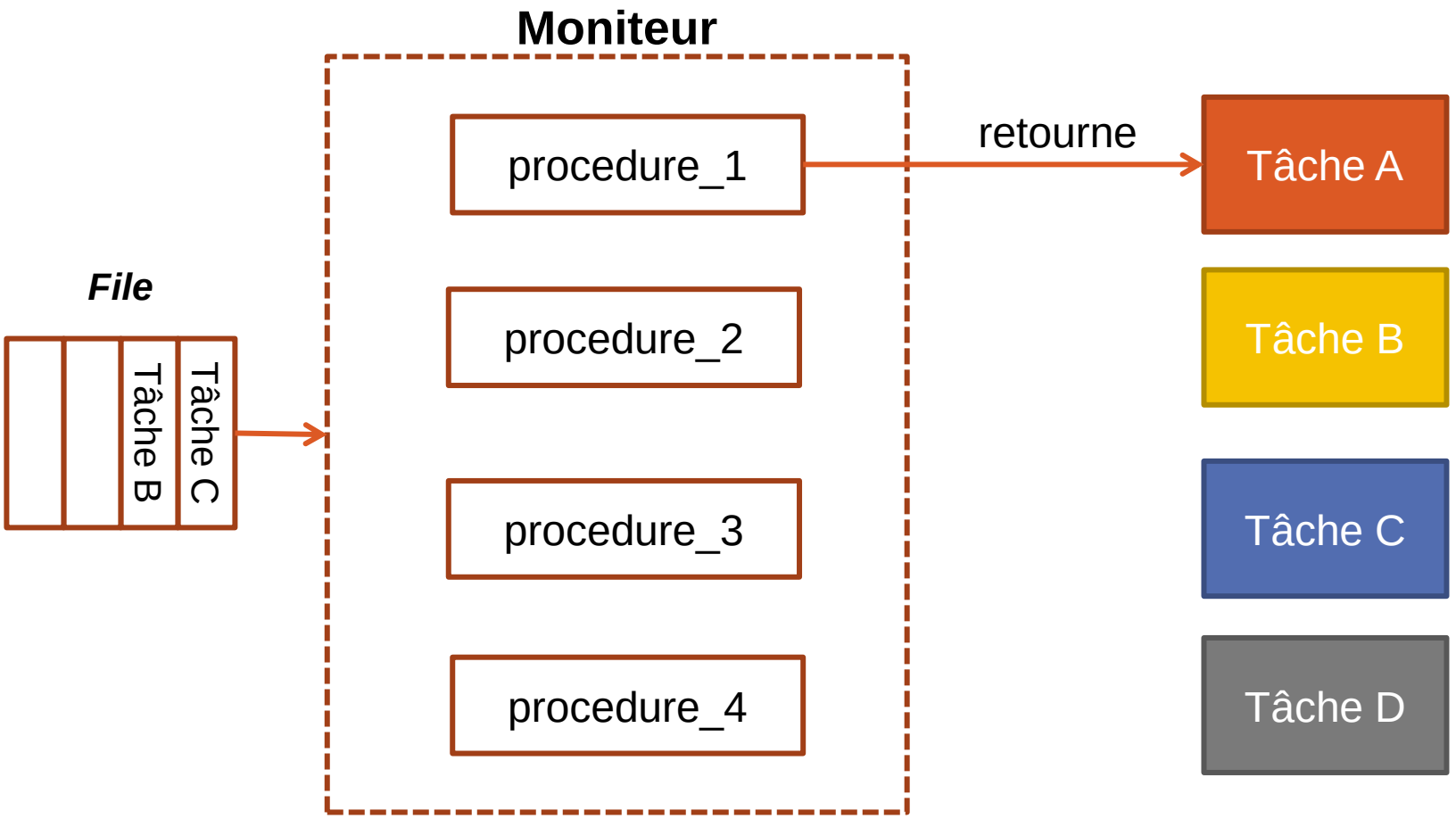
Propriétaire du Moniteur: Tâche A

EXEMPLE D'UTILISATION D'UN MONITEUR



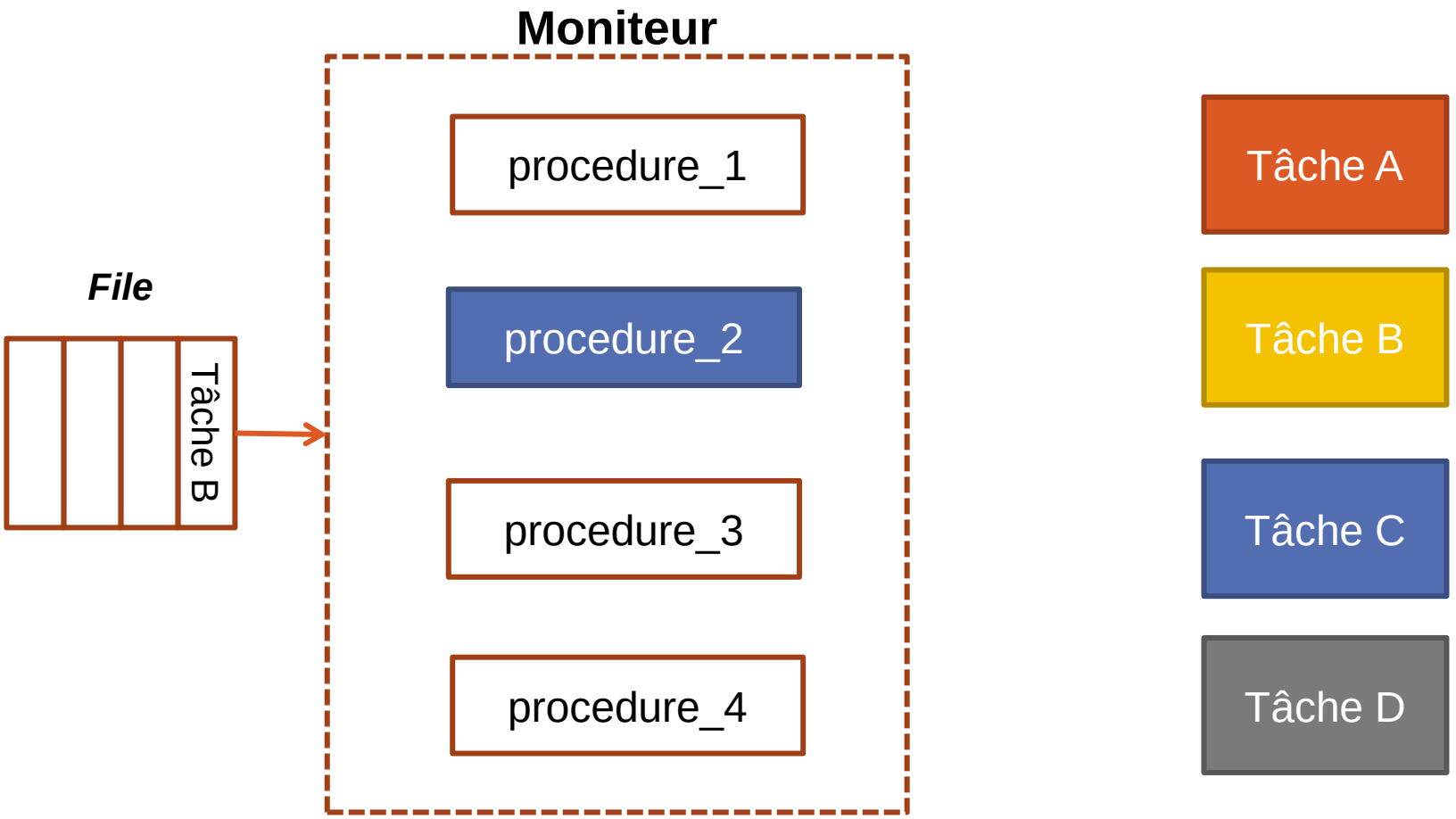
Propriétaire du Moniteur: Tâche A

EXEMPLE D'UTILISATION D'UN MONITEUR



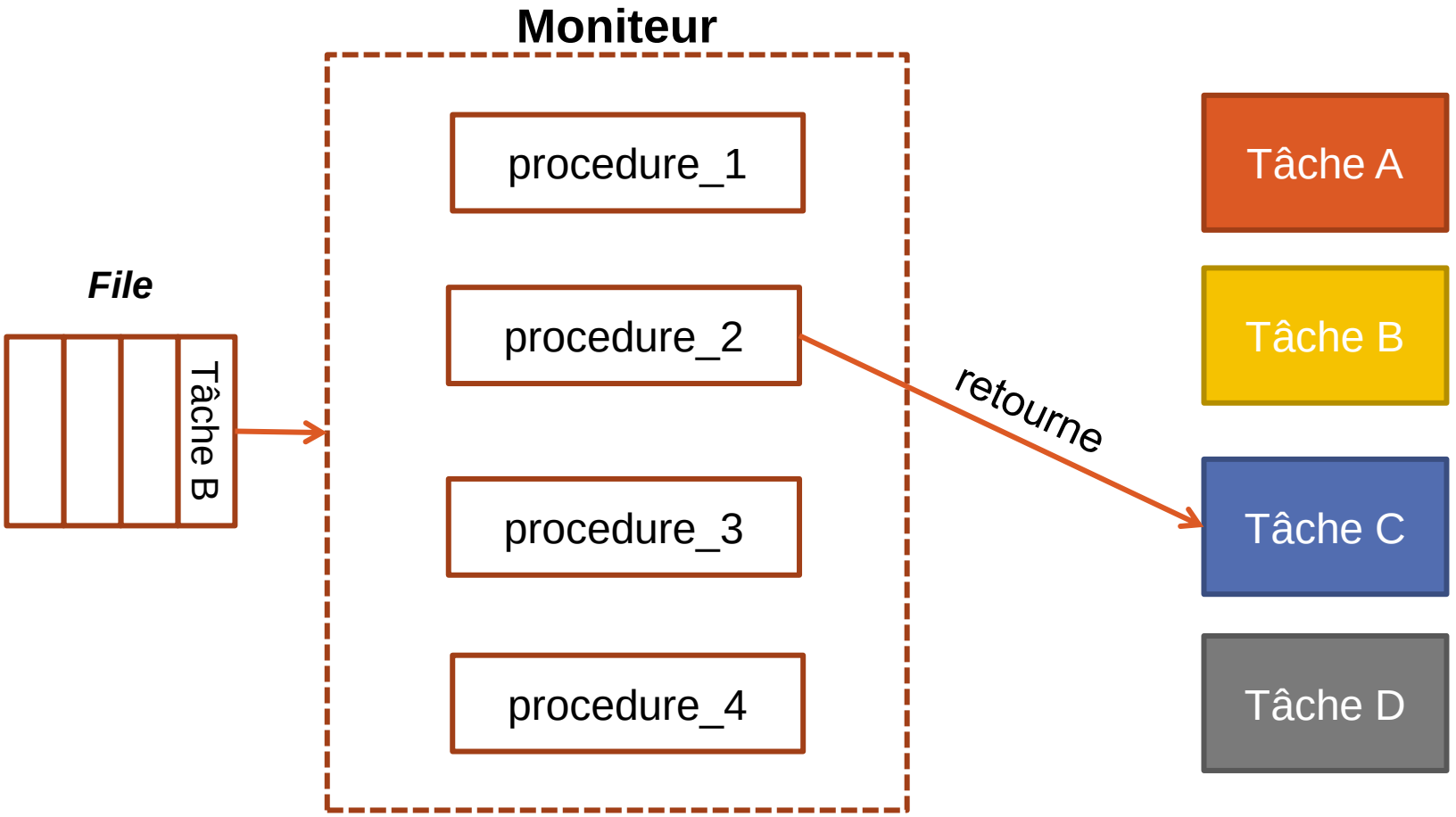
Propriétaire du Moniteur: Aucun

EXEMPLE D'UTILISATION D'UN MONITEUR



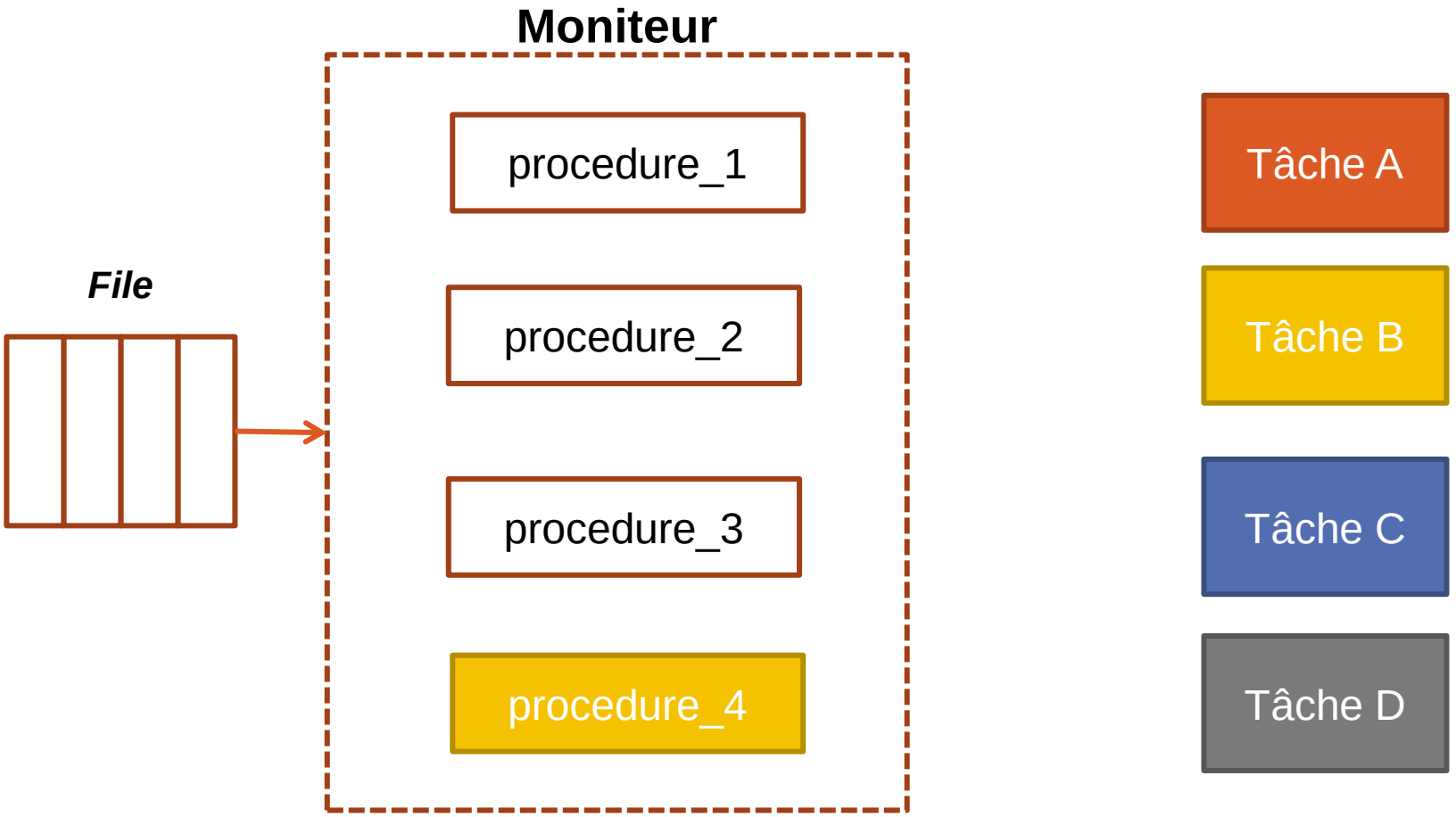
Propriétaire du Moniteur: Tâche C

EXEMPLE D'UTILISATION D'UN MONITEUR



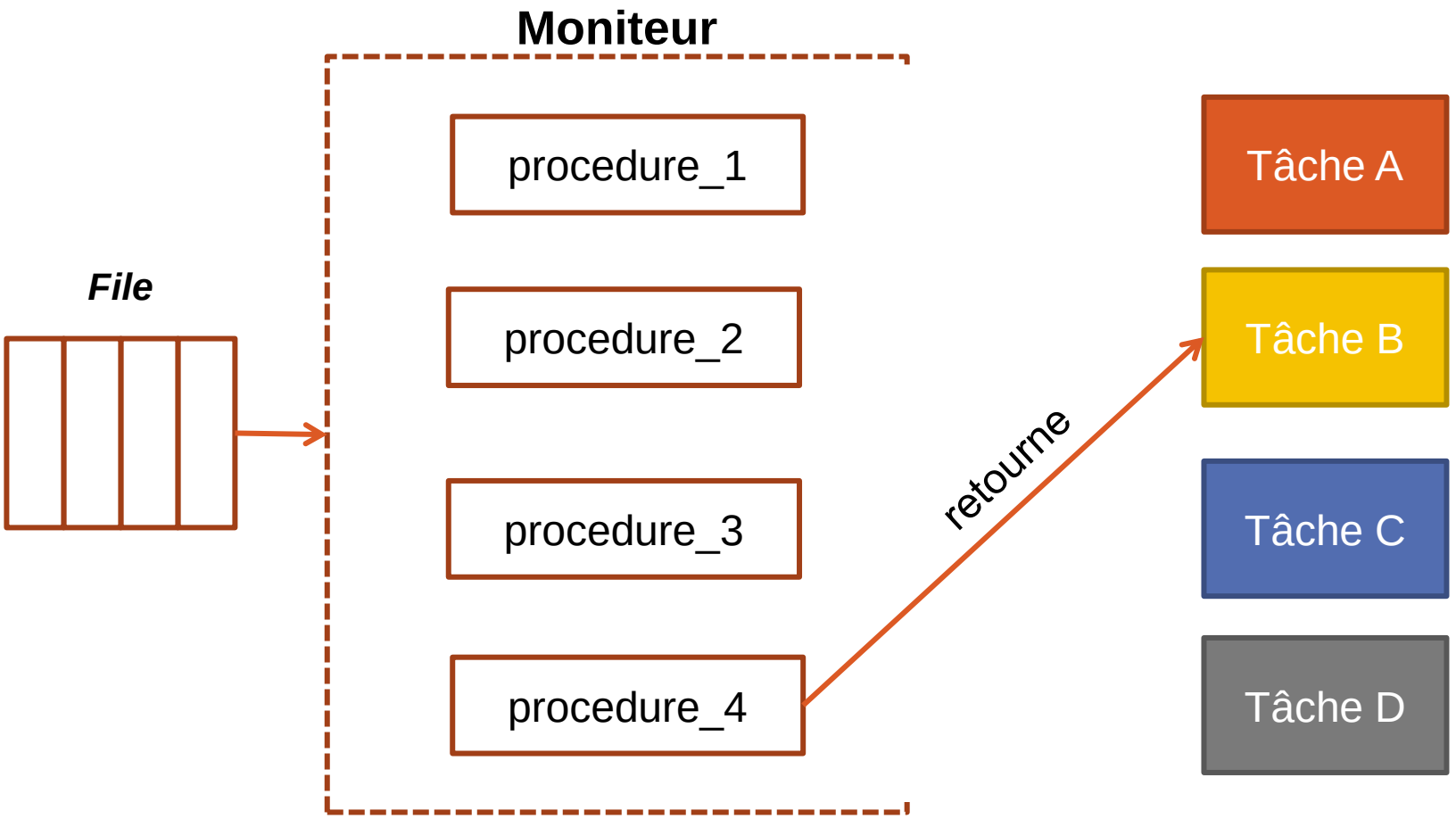
Propriétaire du Moniteur: Aucun

EXEMPLE D'UTILISATION D'UN MONITEUR



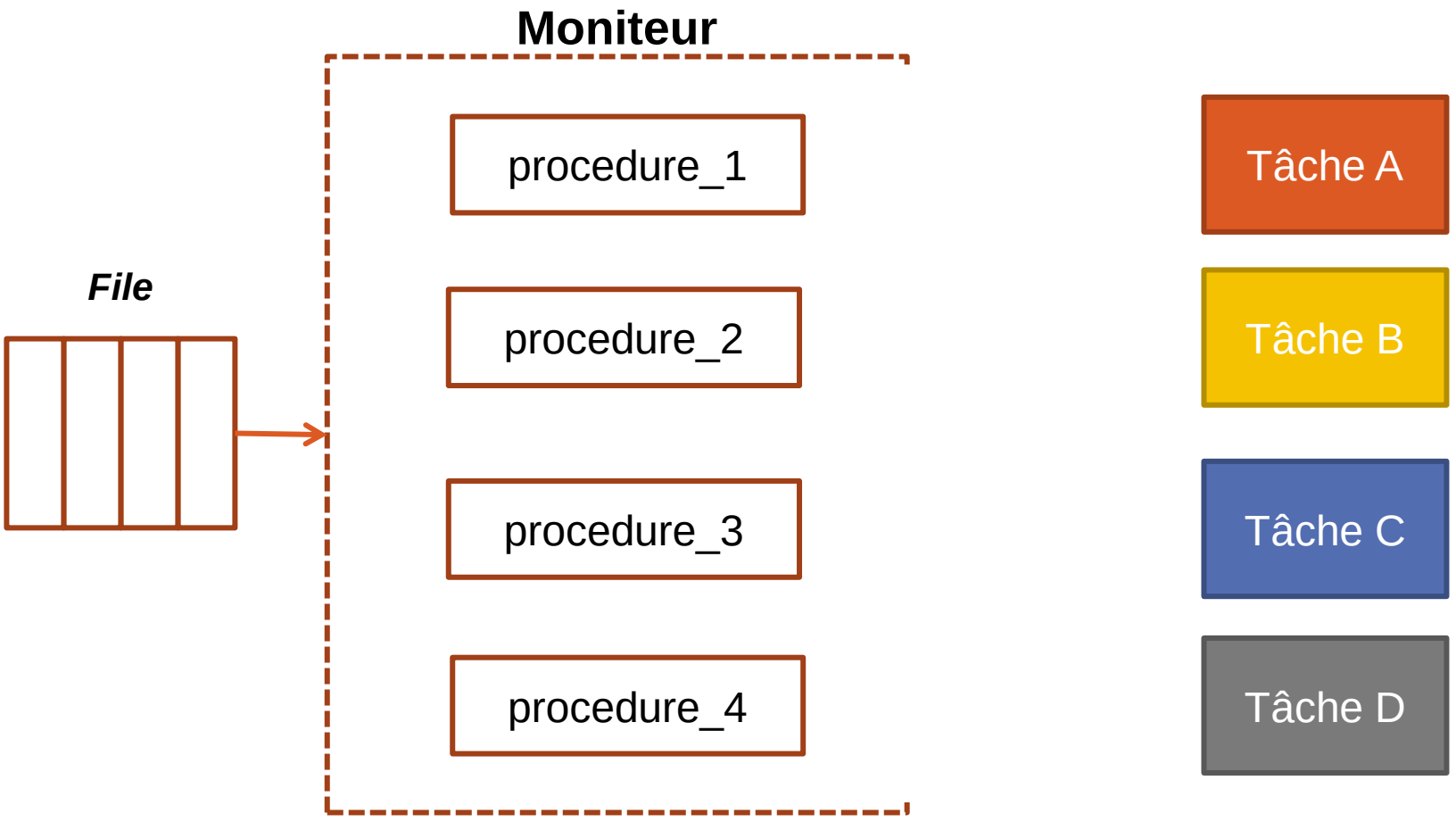
Propriétaire du Moniteur: Tâche B

EXEMPLE D'UTILISATION D'UN MONITEUR



Propriétaire du Moniteur: Aucun

EXEMPLE D'UTILISATION D'UN MONITEUR



Propriétaire du Moniteur: Aucun

EXEMPLE DE COMPTE BANCAIRE

uOttawa
L'Université canadienne
Canada's university

```
MONITOR: Account
          double balance


          procedure double withdraw(amount )
          begin
             balance = balance – amount


             return balance
          end procedure
```

withdraw (amount)
balance = balance - amount

withdraw (amount)

withdraw (amount)

return balance

balance = balance – amount
return balance

balance = balance – amount
return balance

SYNCHRONISATION DE COOPÉRATION

uOttawa
L'Université canadienne
Canada's university

Malgré que l'accès mutuellement exclusif à l'information partagée soit intrinsèque avec un moniteur:

- La coopération entre les processus est toujours la tâche du programmeur

Le programmeur doit garantir qu'un tampon partagé ne subit pas un débordement

VARIABLES DE CONDITION

Les variables de condition fournissent un mécanisme d'attente pour les événements

Les variables de condition supportent trois opérations:

- wait (): relâcher le verrou du moniteur et attendre que la variable de condition soit signalée
- signal(): réveille un thread en attente
- braodcast(): réveiller tous les threads en attente

Chaque variable de condition a sa propre file d'attente

- Une tâche qui attend cette condition est bloquée et son descripteur est stocké dans la file d'attente

EXEMPLE: PRODUCTEUR CONSOMMATEUR

uOttawa
L'Université canadienne
Canada's university

PRODUCTEUR

```
Task Producer
begin
   Loop Forever
      // produce new item
      …
     bufferMonitor.deposit(newItem)
   end
End Task
```

CONSOMMATEUR

```
Task Consumer
begin
   Loop Forever
      newItem = bufferMonitor.fetch()
      // consume new item
      …
end
End Task
```

EXEMPLE: PRODUCTEUR CONSOMMATEUR – MONITEUR VS SÉMAPHORE

Producteur

```
Task Producer
begin
   Loop Forever
      // produce new item
     bufferMonitor.deposit(newItem)
   end
End Task
```

Consommateur

```
Task Consumer
begin
   Loop Forever
      newItem = bufferMonitor.fetch()
      // consume new item
    end
End Task
```

```
task producer;
  loop
  -- produce VALUE --
  wait(emptyspots);      { wait for a space }
  wait(access);          { wait for access }
  DEPOSIT(VALUE);
  release(access);       { relinquish access }
  release(fullspots);    { increase filled spaces }
  end loop;
end producer;
```

```
task consumer;
  loop
  wait(fullspots);       { make sure it is not empty }
  wait(access);          { wait for access }
  FETCH(VALUE);
  release(access);       { relinquish access }
  release(emptyspots);   { increase empty spaces }
  -- consume VALUE --
  end loop
end consumer;
```

EXEMPLE: PRODUCTEUR CONSOMMATEUR

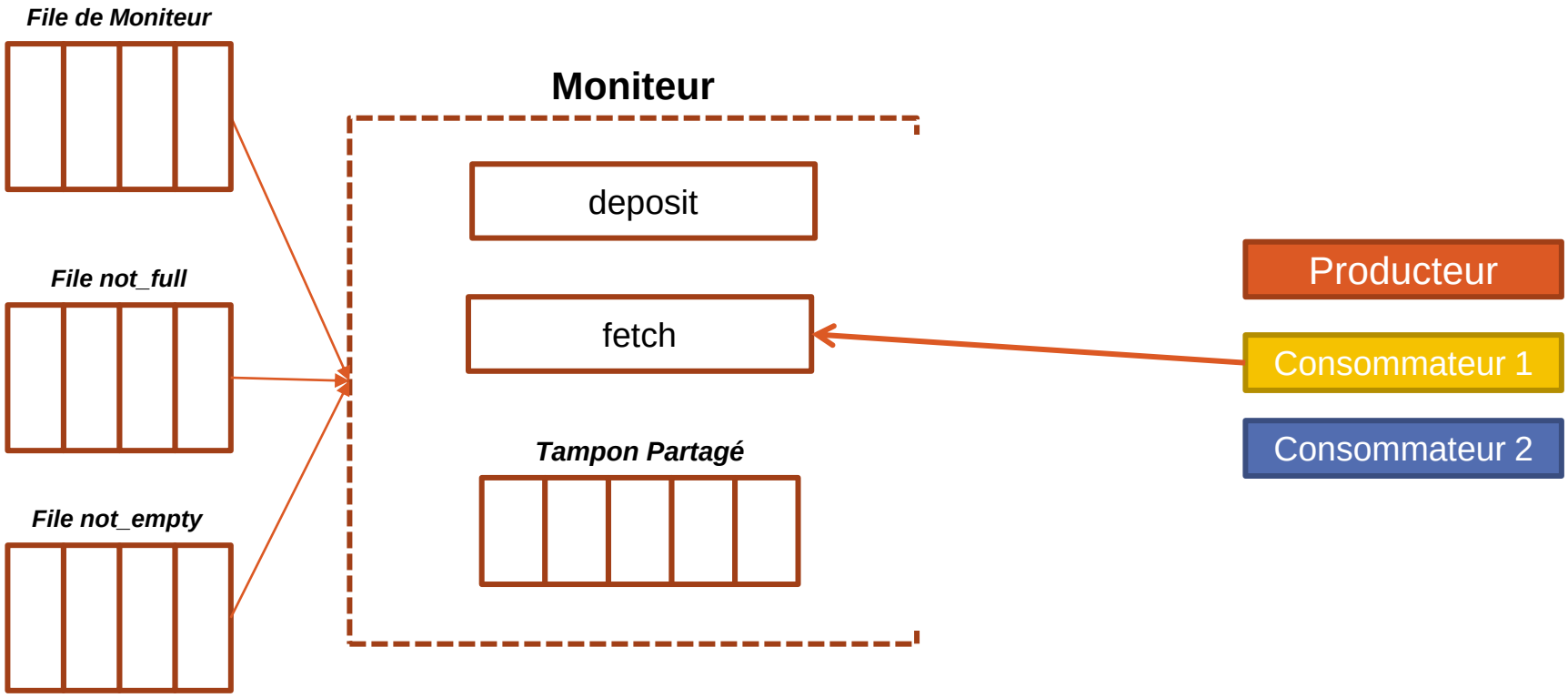
uOttawa
L'Université canadienne
Canada's university

```
MONITOR: BufferMonitor
        const bufferSize = 5
        buffer = array [0.. bufferSize-1]
        next_in = 0, next_out = 0, filled = 0
        condition not_full, not_empty

        procedure void deposit (item )
        begin
           while filled == bufferSize then
              wait(not_full) // block thread and place it in the not_full queue
           end
           buffer[next_in] = item
           next_in = (next_in + 1) mod bufferSize
           filled = filled + 1
           signal(not_empty) // free a task that has been waiting on not_empty
        end procedure
```

EXEMPLE: PRODUCTEUR CONSOMMATEUR

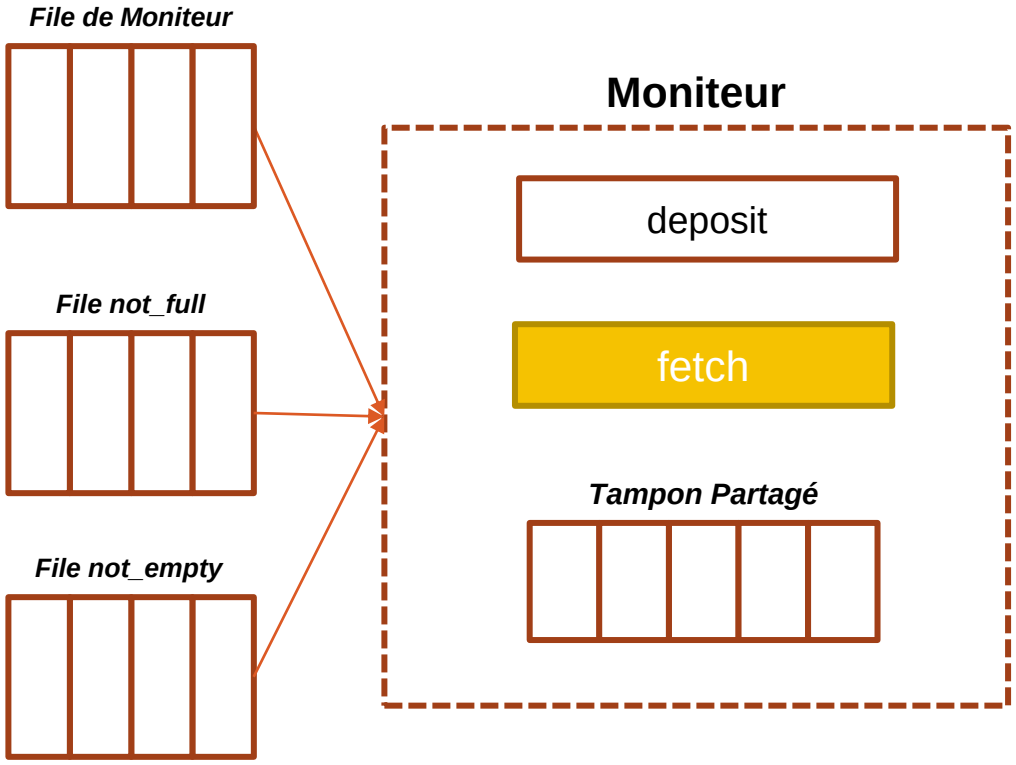

```
procedure Item fetch()
begin
   while filled == 0 then
      wait(not_empty) // block thread and place it in the not_empty queue

   end

   item = buffer[next_out]
   next_out = (next_out + 1) mod bufferSize
   filled = filled - 1

   signal(not_full) // free a task that has been waiting on not_full

   return item
end procedure
```

EXEMPLE: PRODUCTEUR CONSOMMATEUR



Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSOMMATEUR



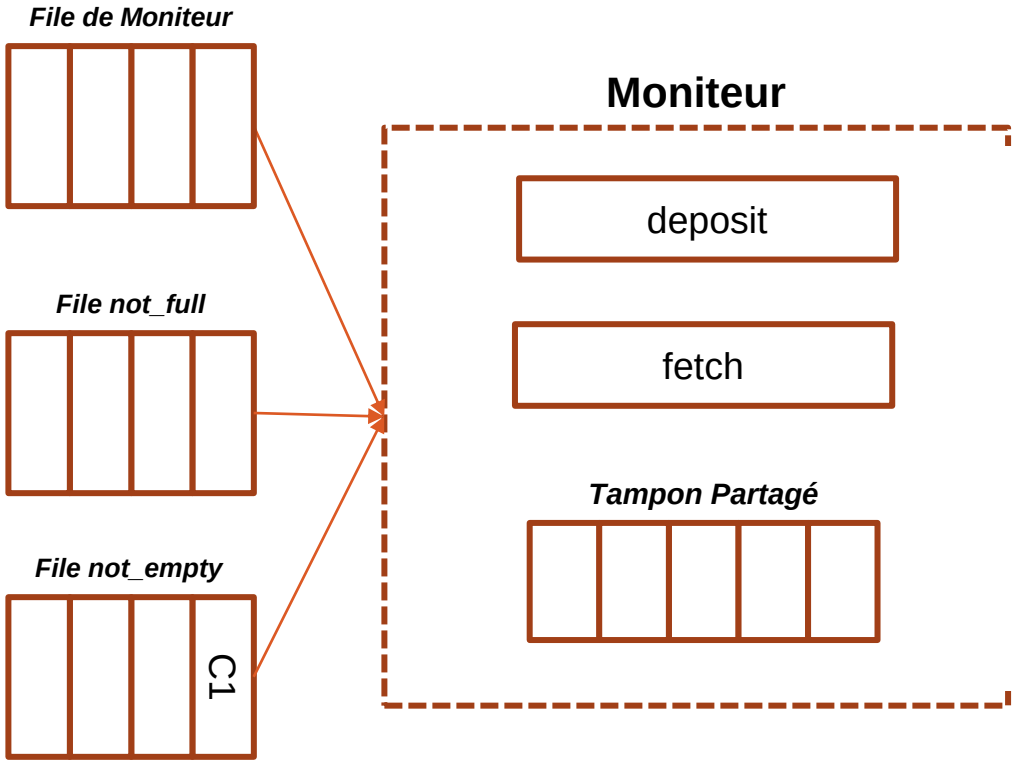
Producteur

Consommateur 1

Consommateur 2

Propriétaire du Moniteur: Consommateur 1

EXEMPLE: PRODUCTEUR CONSOMMATEUR



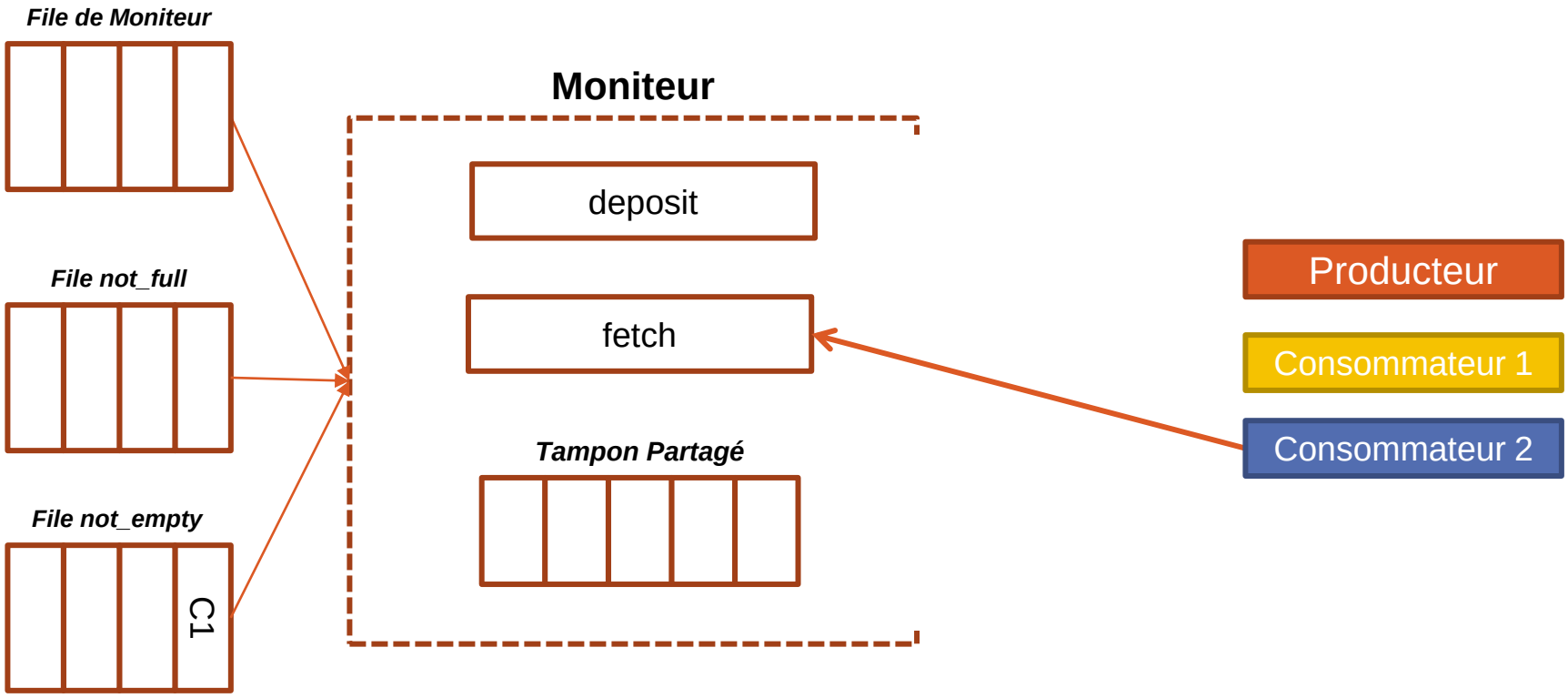
Producteur

Consommateur 1

Consommateur 2

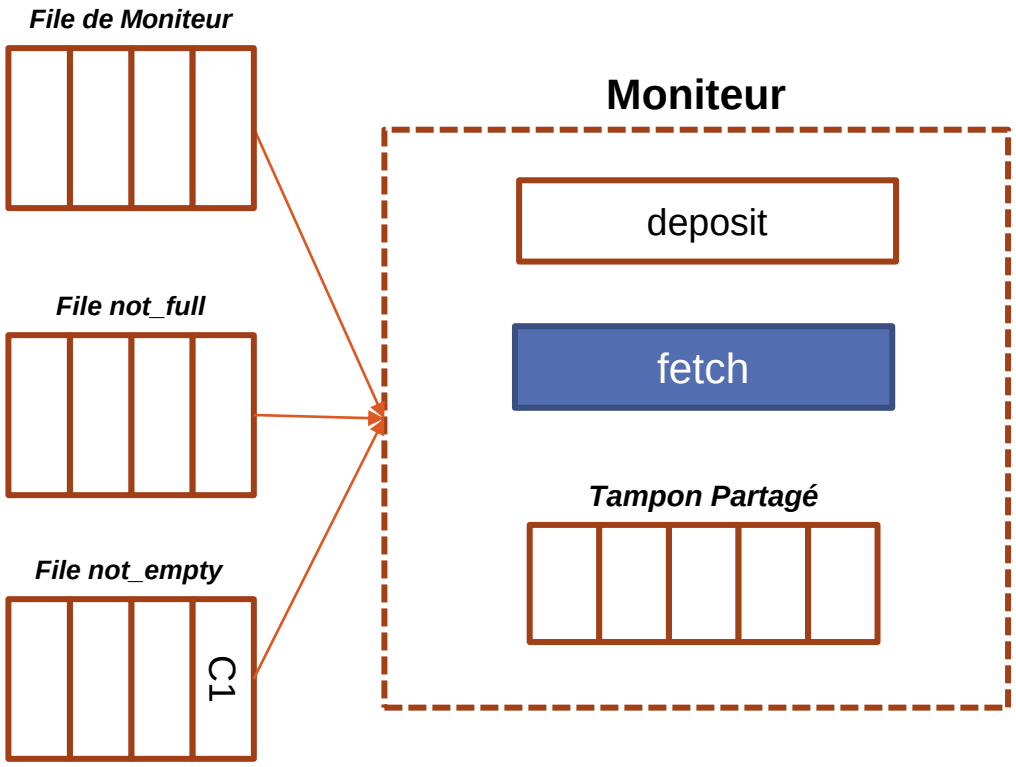
Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSOMMATEUR



Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSOMMATEUR



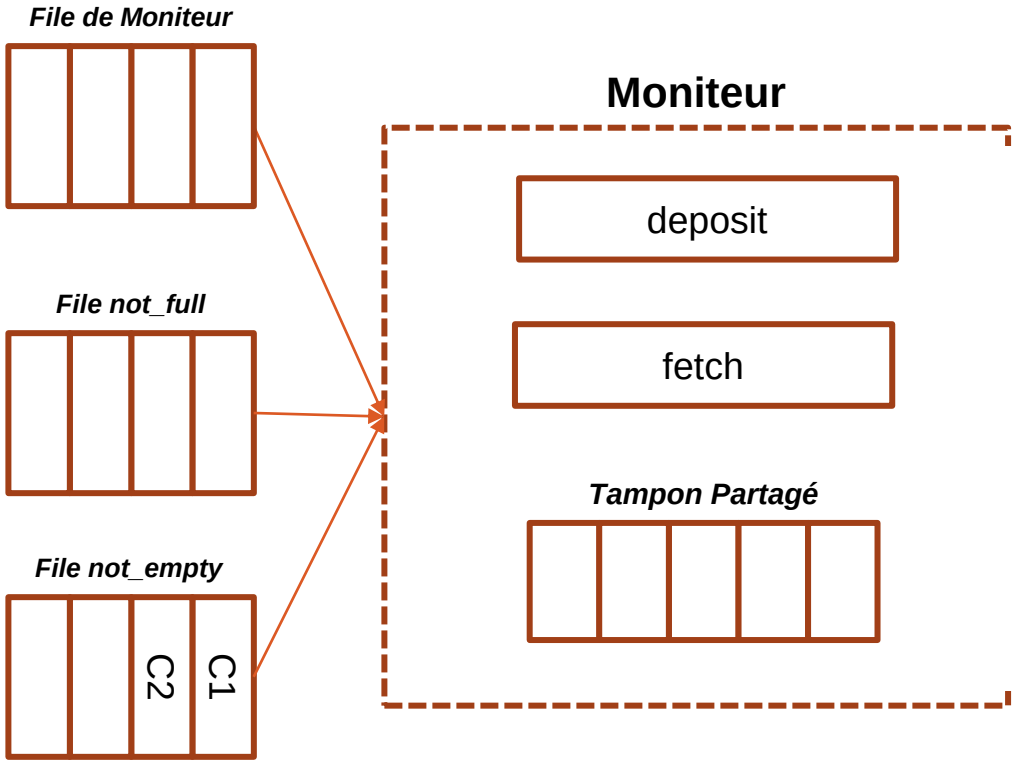
Producteur

Consommateur 1

Consommateur 2

Propriétaire du Moniteur: Consommateur 2

EXEMPLE: PRODUCTEUR CONSOMMATEUR



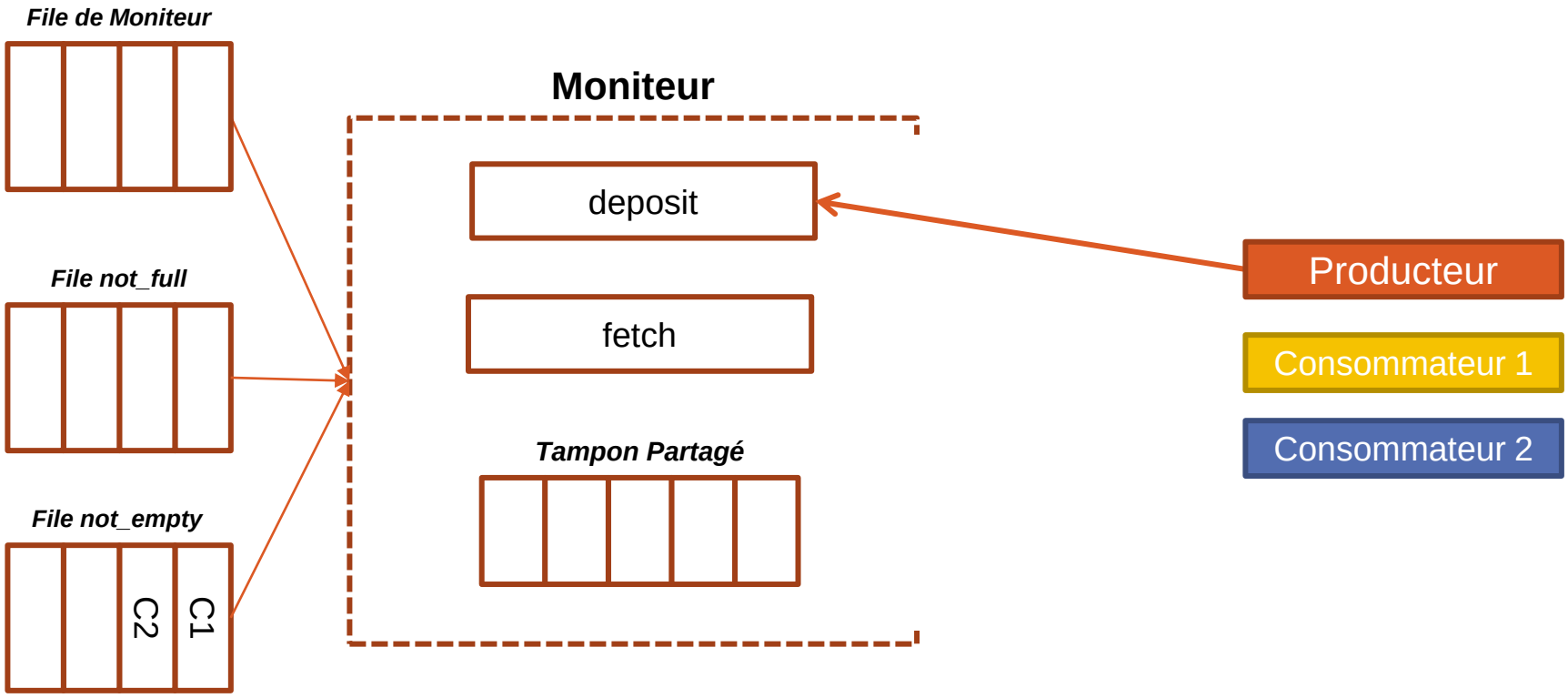
Producteur

Consommateur 1

Consommateur 2

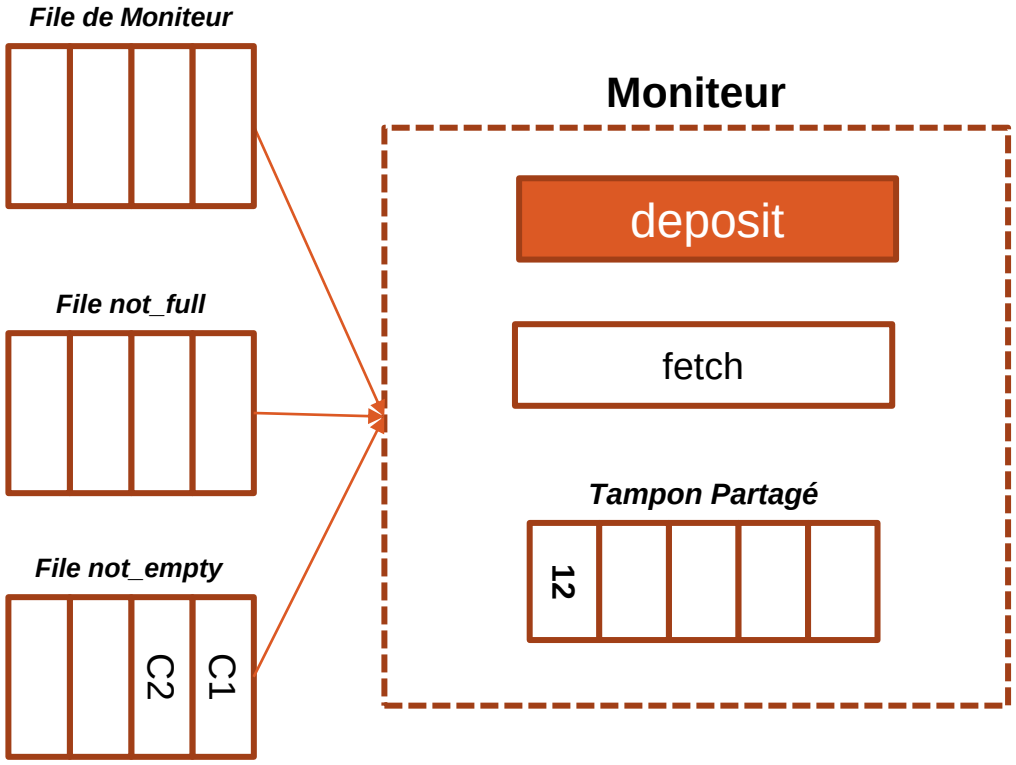
Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSOMMATEUR



Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSOMMATEUR



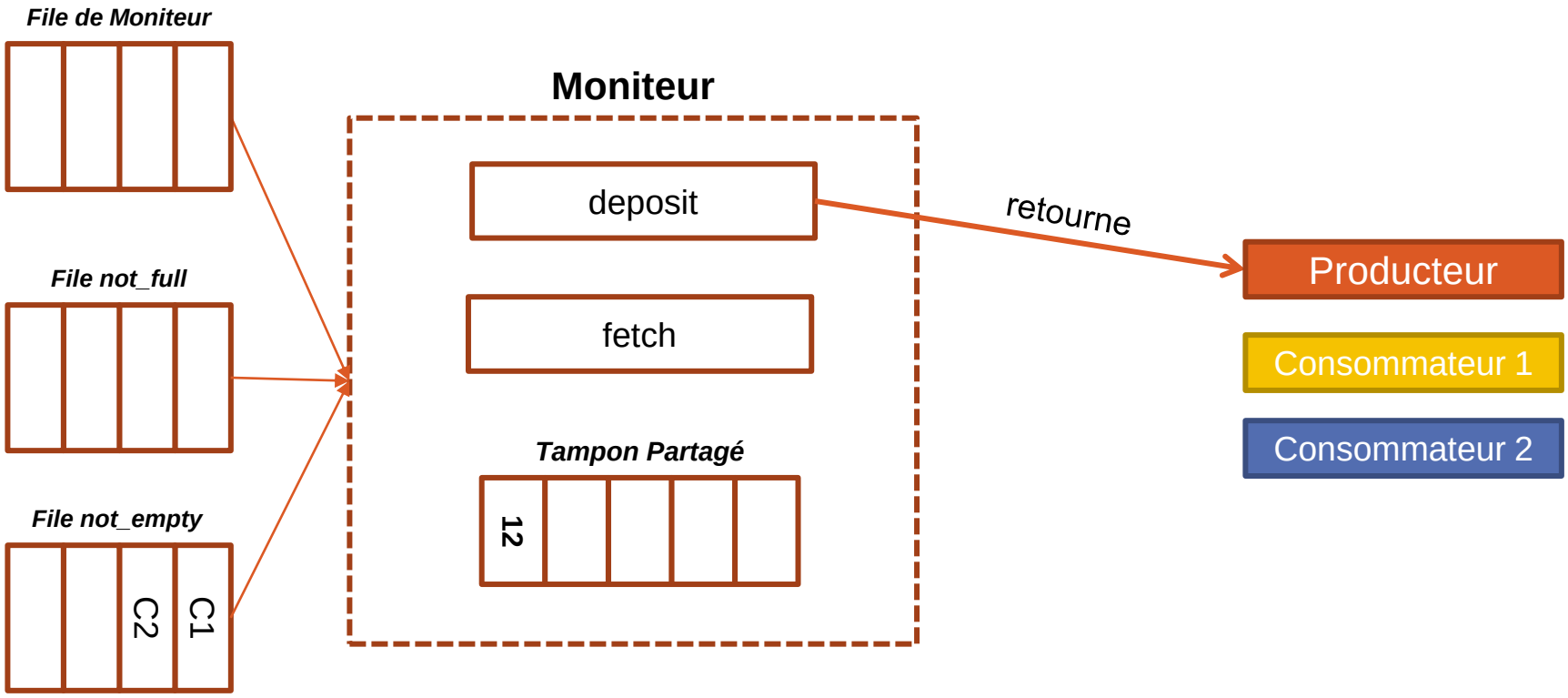
Producteur

Consommateur 1

Consommateur 2

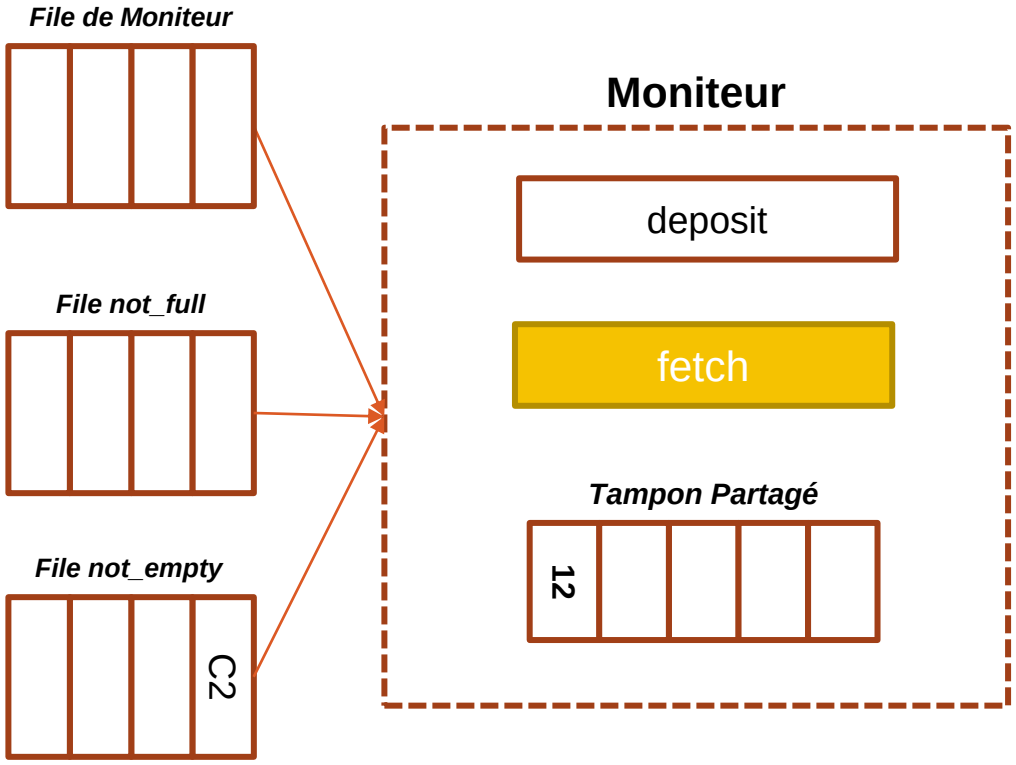
Propriétaire du Moniteur: Producteur

EXEMPLE: PRODUCTEUR CONSOMMATEUR



Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSOMMATEUR



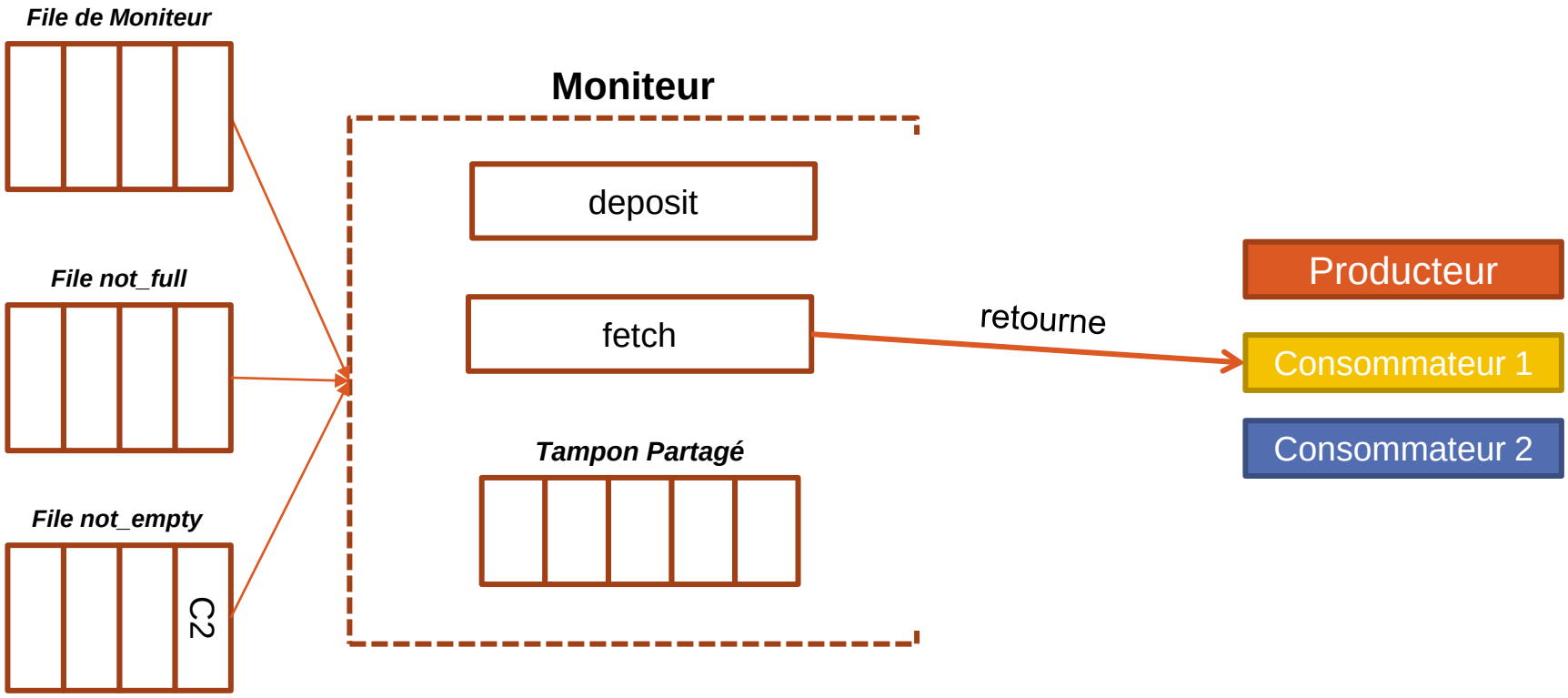
Producteur

Consommateur 1

Consommateur 2

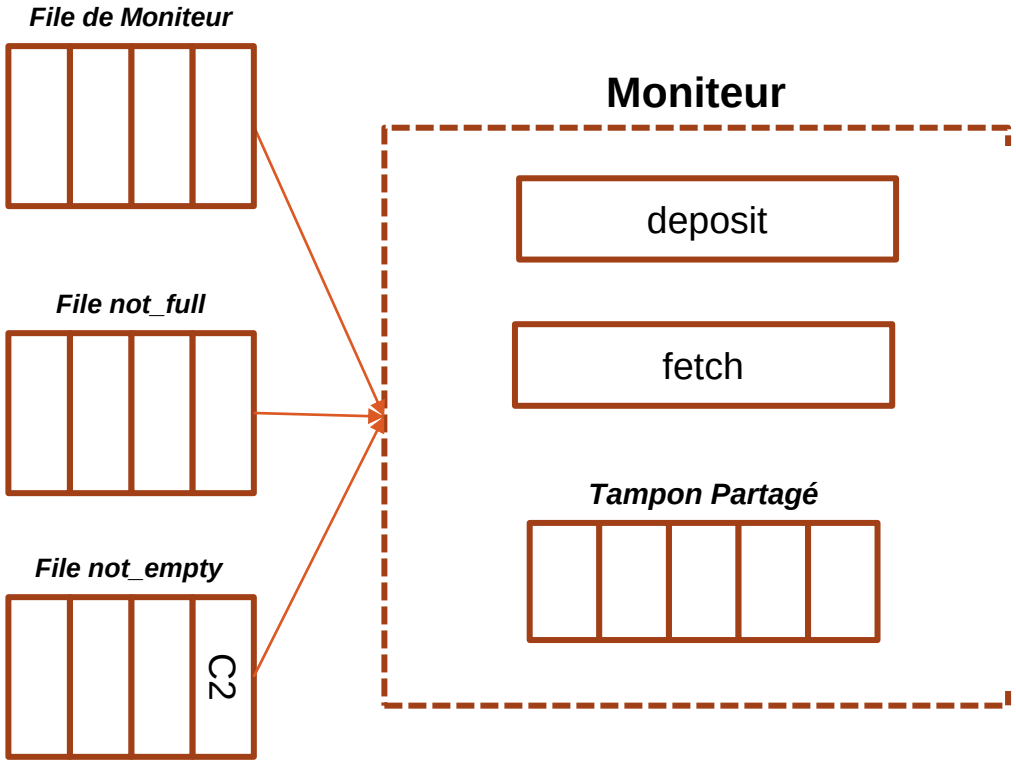
Propriétaire du Moniteur: Consommateur 1

EXEMPLE: PRODUCTEUR CONSOMMATEUR



Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSOMMATEUR



Producteur

Consommateur 1

Consommateur 2

Propriétaire du Moniteur: Aucun

LES VARIABLES DE CONDITION NE SONT PAS DES SÉMAPHORES

uOttawa
L'Université canadienne
Canada's university

Les variables de condition ne sont pas équivalentes aux sémaphores

- Bien que les opérations ont des noms similaires, elles ont des interprétations différentes

Pour les variables de condition des moniteurs:

- L'accès au moniteur est contrôlé par un verou
- wait () bloque le thread appelant et lui cause de lacher le verrou
  - Pour appeler wait (), le thread doit être dans le moniteur
- signal () provoque le réveil d'un thread en attente
  - S'il n'y a pas de thread en attente, le signal est ignoré

Pour les sémaphores:

- wait () bloque le thread si la section critique est occupée
- release () augmente le compteur du sémaphore (à moins que ce soit un sémaphore binaire, dans ce cas, il le fixe à '1')

MERCI!

QUESTIONS?