Tampon producteurs-consommateurs

- **On considère un tampon circulaire producteurs-consommateurs de N cases**
 - Pour simplifier, par la suite, on choisira $N=1$
- **On considère M threads producteurs et M threads consommateurs**
 - 1 seule production par producteur
 - 1 seule consommation par consommateur
 - Pour simplifier, par la suite, on choisira $M=3$
- **On suppose que la synchronisation des accès au tampon est gérée avec des moniteurs Java, de la façon suivante :**
 - On utilise la variable de condition associée à l'objet tampon pour bloquer les producteurs (lorsque le tampon est plein) et pour bloquer les consommateurs (lorsque le tampon est vide)
 - Lorsqu'un producteur a fini de déposer une donnée, il invoque notify sur le tampon pour débloquent un éventuel consommateur bloqué en attente d'une donnée
 - Lorsqu'un consommateur a fini de déposer une donnée, il invoque notify sur le tampon pour débloquent un éventuel producteur bloqué en attente d'un emplacement libre pour produire

Considérons la séquence d'exécution suivante ...

Initialement : tampon vide

P1 produit (on a 1 élément)

P2 essaye de produire et se bloque en attente d'une place

P3 essaye de produire et se bloque en attente d'une place

C1 consomme (le tampon devient vide) et le notify choisit de débloquent P2, qui ne prend pas la main tout de suite

C2 essaye de consommer et se bloque en attente d'un élément

C3 essaye de consommer et se bloque en attente d'un élément

(à ce stade là, dans la file d'attente, on a 1 producteur et 2 consommateurs)

P2 reprend la main, produit (le tampon devient plein), et le notify choisit de débloquent P3

(on a débloquent un producteur alors que l'on pensait débloquent un consommateur)

P3 reprend la main, réessaye de produire et se bloque à nouveau (tampon plein)

Au final, il reste un tampon plein et trois threads bloqués ... alors que l'on avait un équilibre entre producteurs et consommateurs (on devrait finir avec le tampon vide et chaque thread ayant réussi à produire/consommer)

Conclusion

- **N'utiliser qu'une seule file d'attente (variable de condition) à la fois pour les producteurs et les consommateurs est délicat**
- **Deux approches possibles pour éviter le problème :**
 - Garder une file d'attente commune mais réveiller systématiquement tous les threads après chaque consommation/production (pour être sûr que l'on réveille bien tous les threads)
 - Approche potentiellement inefficace (car la plupart des threads débloqués à chaque fois devront se bloquer à nouveau)
 - Utiliser deux files d'attentes disjointes : une pour les producteurs et une pour les consommateurs
 - Comment faire en Java ? (voir TP)