```
/** PCO 2015 YTA Buffer2Conso. Tampon simple, 1 producteur - 2 consommateurs.
 * --- 2eme solution --- Objectif: parfaitement fonctionnel.
 * Commentaire: optimisation ok, beaucoup mieux, générique (évidemment, plus compli
template<typename T> class Buffer2Conso : public AbstractBuffer<T> {
public:
    Buffer2Conso();
    virtual ~Buffer2Conso();
    virtual void put(T item);
    virtual T get(void);
private:
    T element;
    QSemaphore* waitProducteur;
    QSemaphore* waitConsommateur;
    QSemaphore* mutex; // protège les variables ci-dessous
    bool full; // indique l'état du tampon: plein ou vide.
    int nbWaitProducteur; // nombre de producteurs en attente
    int nbWaitConsommateur; // nombre de consommateurs en attente.
    int nbConsommation; // nombre de consommation effective du tampon.
}

Buffer2Conso::Buffer2Conso() {
    waitProducteur = new QSemaphore();
    waitConsommateur = new QSemaphore();
    mutex = new QSemaphore();
    mutex->release(); // libre
    full = false; // vide
}

Buffer2Conso::~Buffer2Conso() {
    delete waitProducteur;
    delete waitConsommateur;
    delete mutex;
}

void Buffer2Conso::put(T item) {
    mutex->acquire();
    if (full) { // prélude: si c'est déjà plein: on se met en attente.
        nbWaitProducteur++;
        mutex->release(); // liberation du mutex.
        waitProducteur->acquire();
        // pas besoin de mutex.acquire: transmission de mutex.
    }
    // critique: poser l'element.
    element = item;
    // post-lude: on libère un consommateur s'il y en a en attente
    if (nbWaitConsommateur > 0) {
        nbWaitConsommateur--;
        waitConsommateur->release(); // transmission de mutex
    } else {
        full = true;
        mutex->release(); // sinon on libère la ressource
    }
}

```

```
55 T Buffer2Conso::get(void) {
56     T item;
57     mutex->acquire();
58     if(!full) { // prélude: s'il n'y a rien à consommer: on se met en attente.
59         nbWaitConsommateur++;
60         mutex->release();
61         waitConsommateur->acquire();
62         // pas besoin de mutex->acquire: transmission de mutex.
63     }
64     // section critique: récupérer l'élément.
65     item = element;
66     nbConsommation++; // il y a eu consommation!
67     // post-lude: 1ere version (pas très claire)
68     if (nbConsommation == 2 && nbWaitProducteur > 0) {
69         nbConsommation = 0;
70         nbWaitProducteur--;
71         waitProducteur->release();  // transmission de mutex
72     } else if (nbConsommation < 2) {
73         if (nbWaitConsommateur > 0) {
74             nbWaitConsommateur--;
75             waitConsommateur->release();  // transmission de mutex
76         } else {
77             mutex->release();
78         }
79     } else {
80         nbConsommation = 0;
81         full = false;
82         mutex->release();
83     }
84     // post-lude: plus simple et plus clair
85     if (nbConsommation == 2) {
86         nbConsommation = 0;
87         // il y a eu 2 consommations: on peut libérer un producteur
88         // s'il n'y en a pas, on libère la ressource et marque le tampon vide.
89         if (nbWaitProducteur > 0) {
90             nbWaitProducteur--;
91             waitProducteur->release();  // transmission de mutex
92         } else {
93             full = false;
94             mutex->release();
95         }
96     } else {
97         // pas encore 2 consommations: on libère un autre consommateur en attente
98         if (nbWaitConsommateur > 0) {
99             nbWaitConsommateur--;
100            waitConsommateur->release();  // transmission de mutex
101        } else {
102            mutex->release(); // sinon, on libère la ressource.
103        }
104    }
105    // on retourne l'élément
106    return item;
107 }
```