
SYSC 3303 Real-Time Concurrent Systems

Synchronizing Java Threads: notify() vs. notifyAll() Example

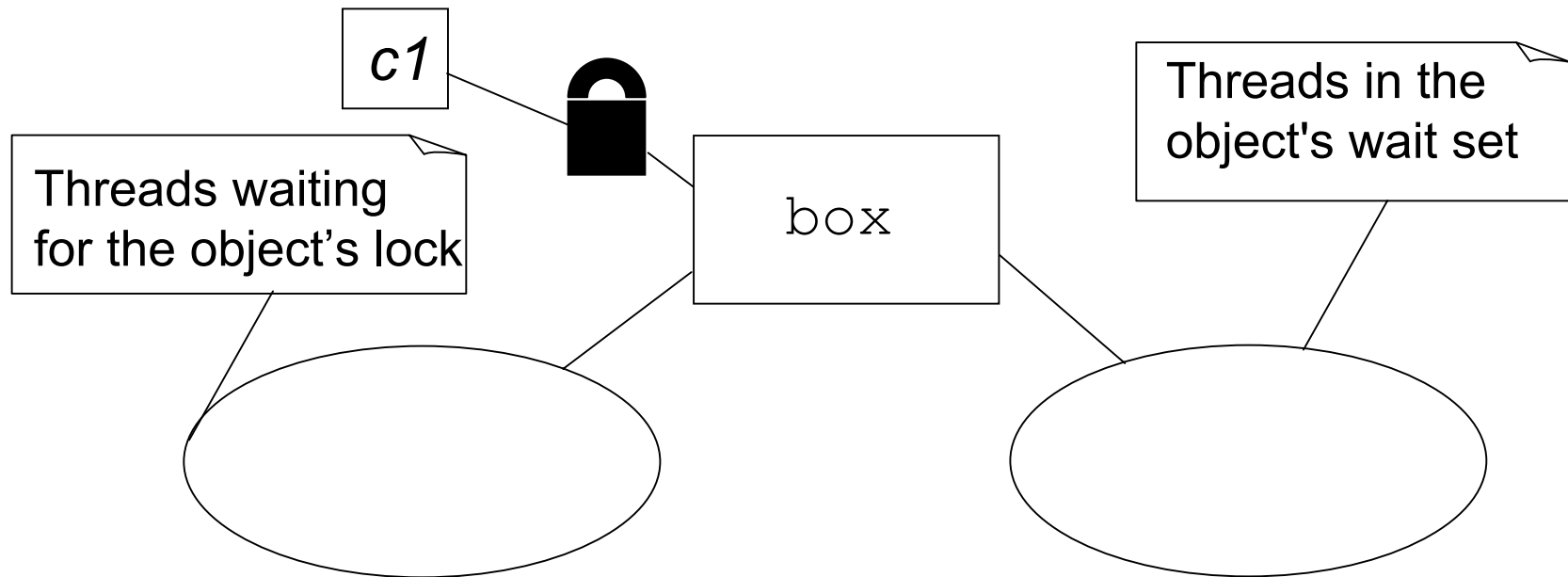
- Copyright © 2003 D.L. Bailey and 2007 L.S.Marshall, Systems and Computer Engineering, Carleton University
- revised July 9th, 2007

Producer/Consumer Execution using notify() instead of notifyAll()

- Suppose the `Box` is empty, and consumer threads `c1` and `c2` invoke `get()`

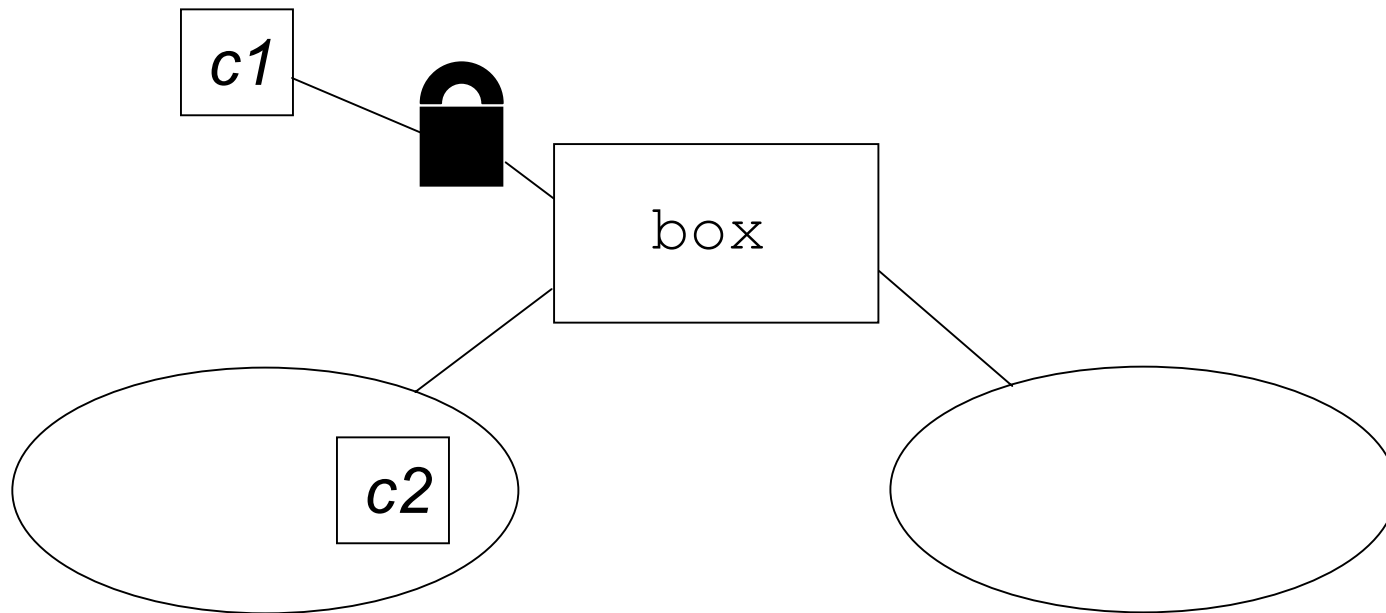
Producer/Consumer Execution using notify()

- *c1* gets the lock



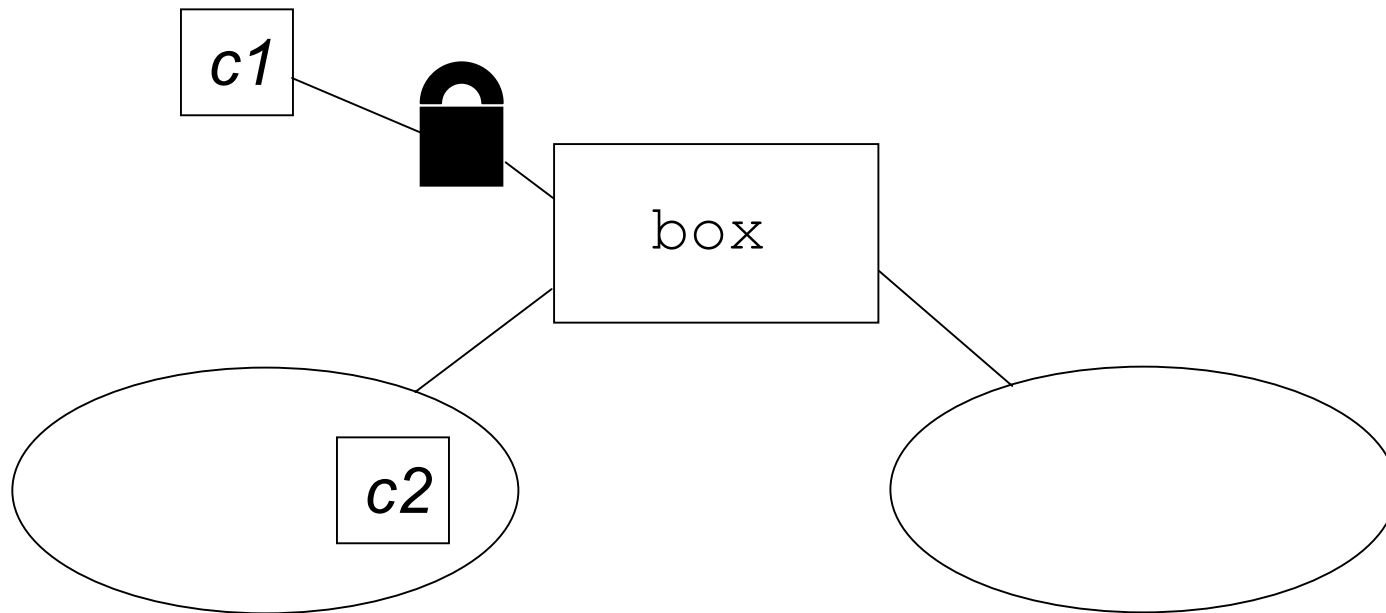
Producer/Consumer Execution using notify()

- *c2* arrives while *c1* has the lock, so *c2* blocks



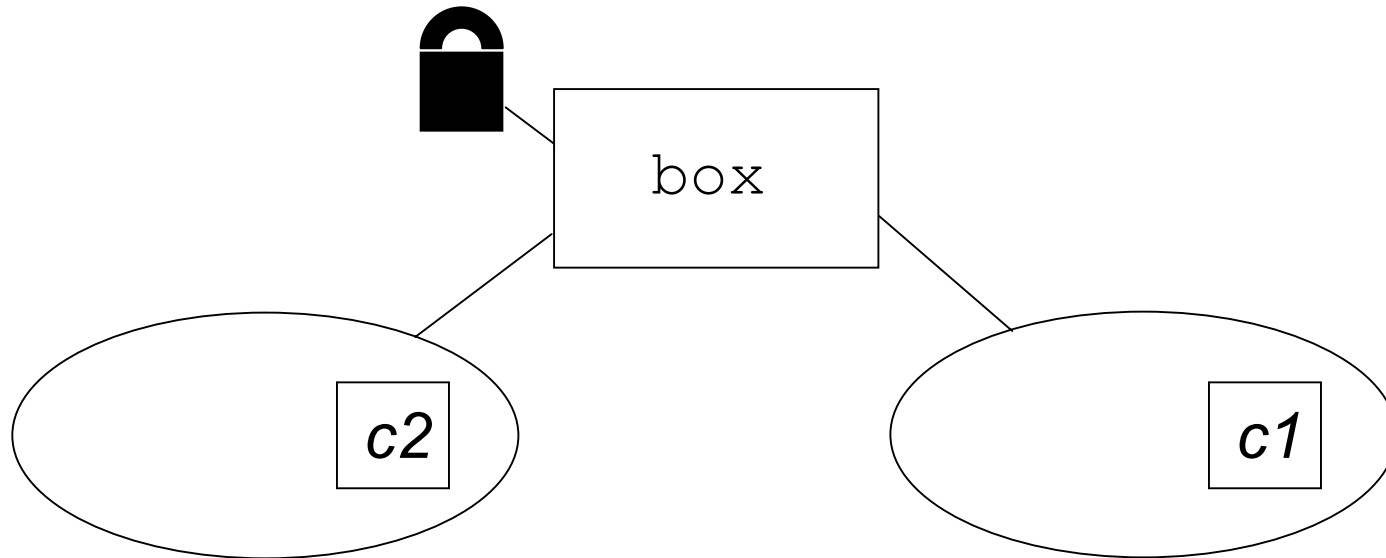
Producer/Consumer Execution using notify()

- *c1* starts executing `get()`



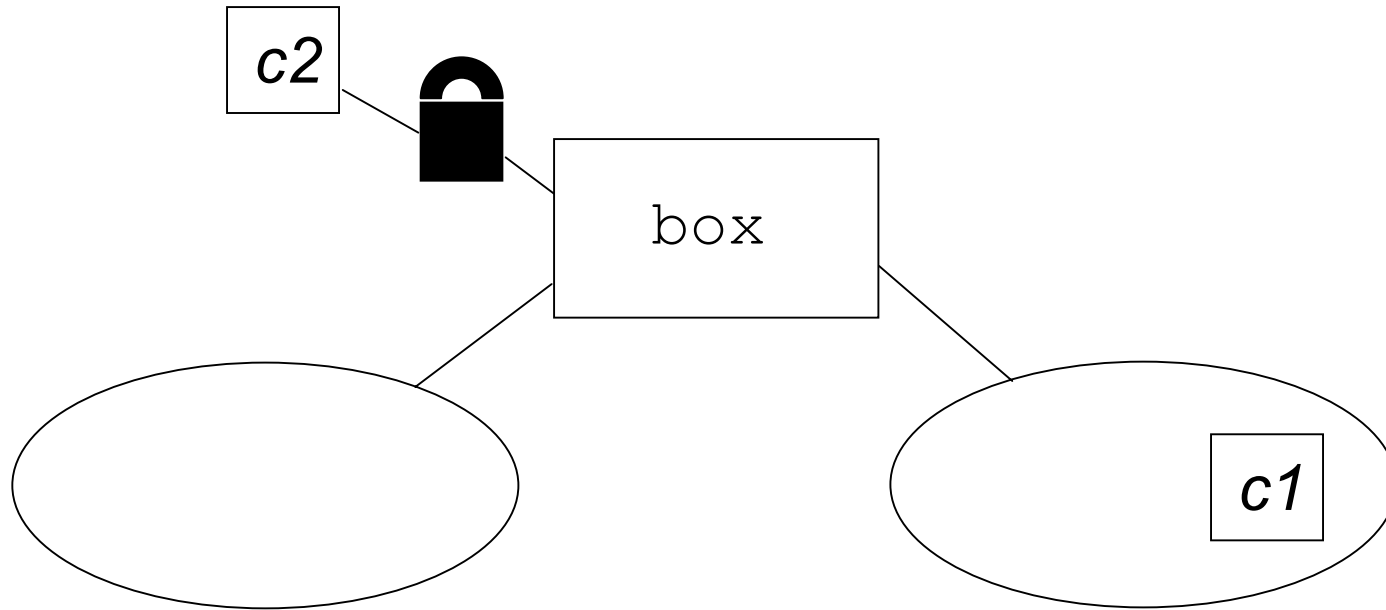
Producer/Consumer Execution using notify()

- *c1* enters the `while` loop body (`empty` is `true`)
 - it invokes `wait()`, releases the object's lock, is placed in the wait set and blocks



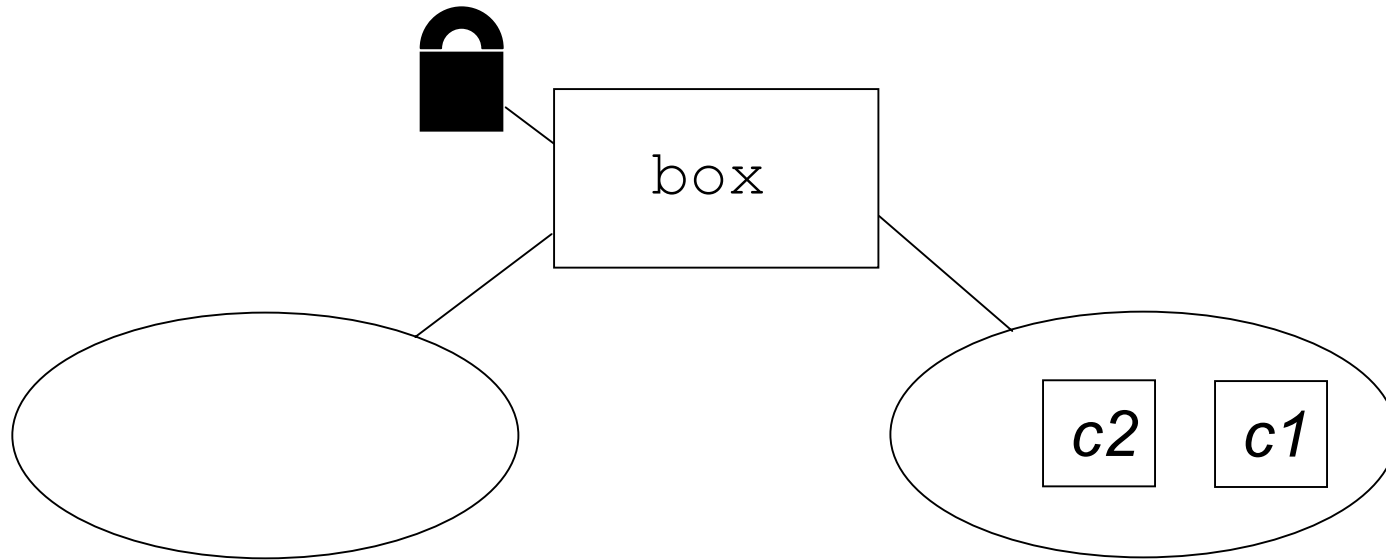
Producer/Consumer Execution using notify()

- The JVM grants the lock to *c2*, which starts executing `get()`



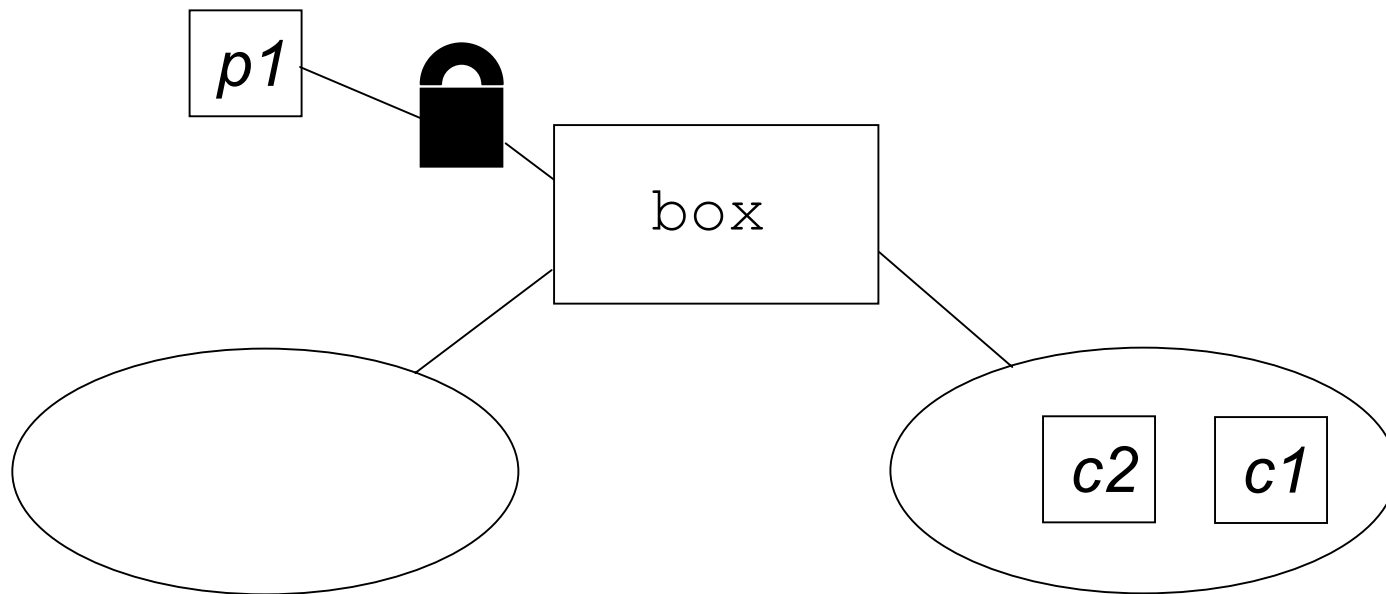
Producer/Consumer Execution using notify()

- `c2` enters the `while` loop body (`empty` is `true`)
 - it invokes `wait()`, releases the object's lock, is placed in the wait set and blocks



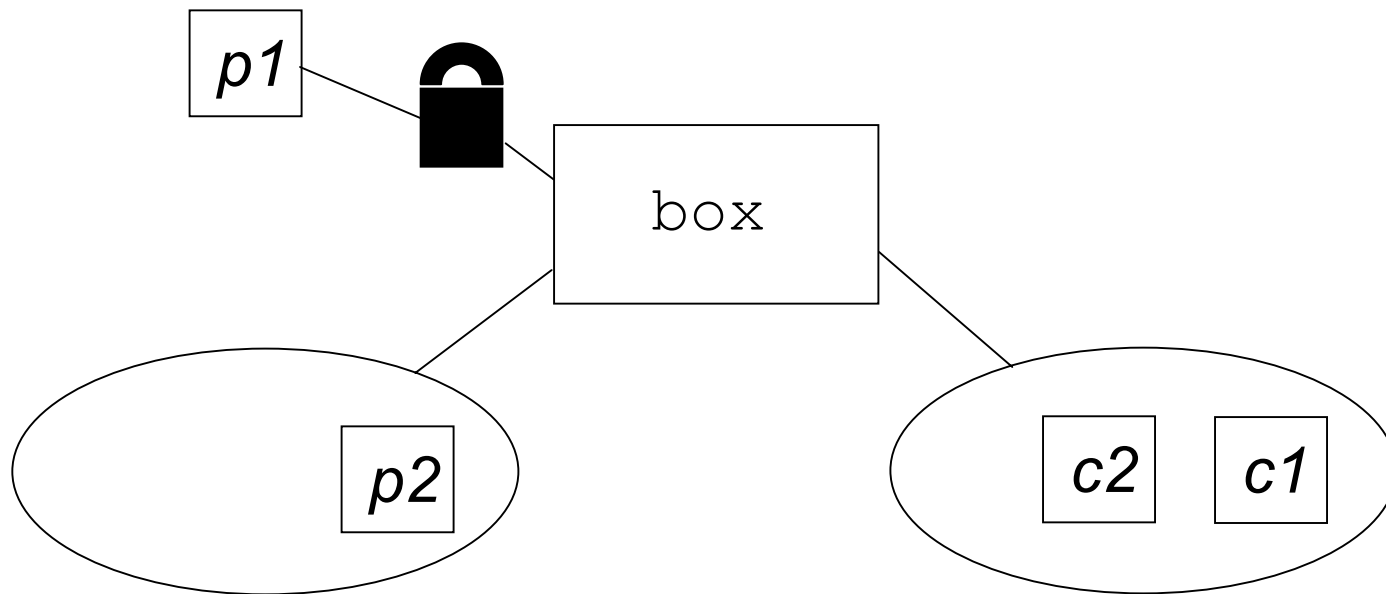
Producer/Consumer Execution using notify()

- Producer *p1* invokes `put()`, obtains the object's lock, and starts executing `put()`
- `empty` is `true`, so it doesn't invoke `wait()`



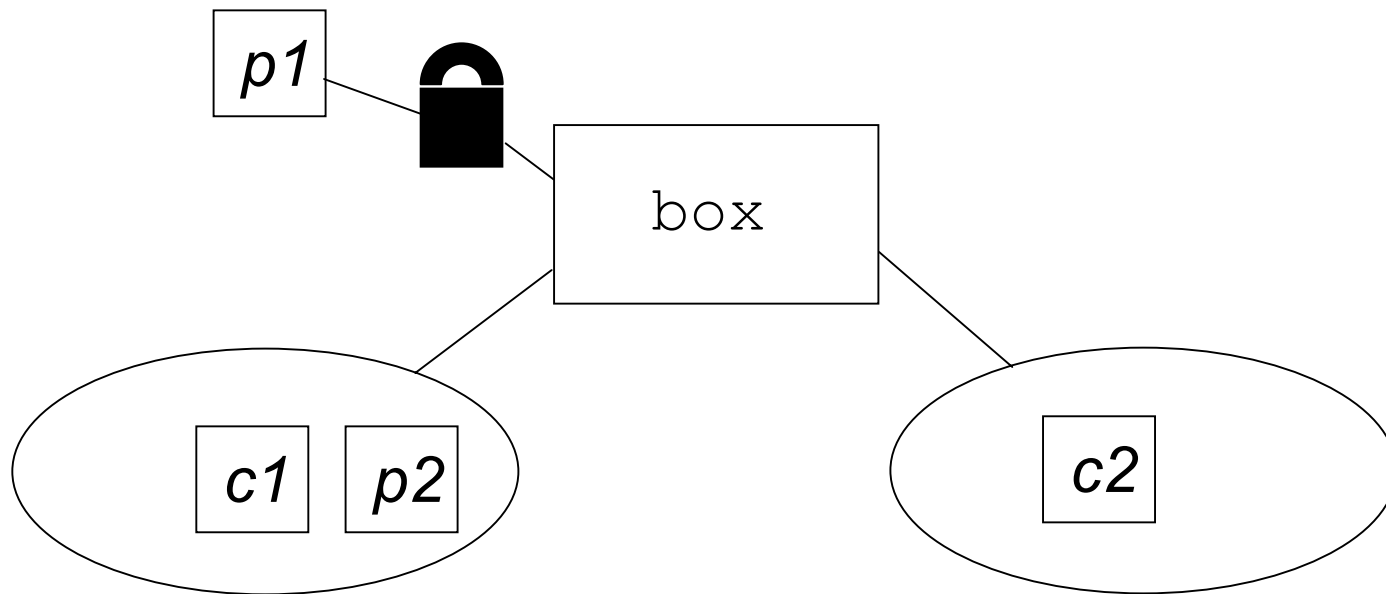
Producer/Consumer Execution using notify()

- Producer *p2* invokes `put()` and blocks until it can obtain the lock



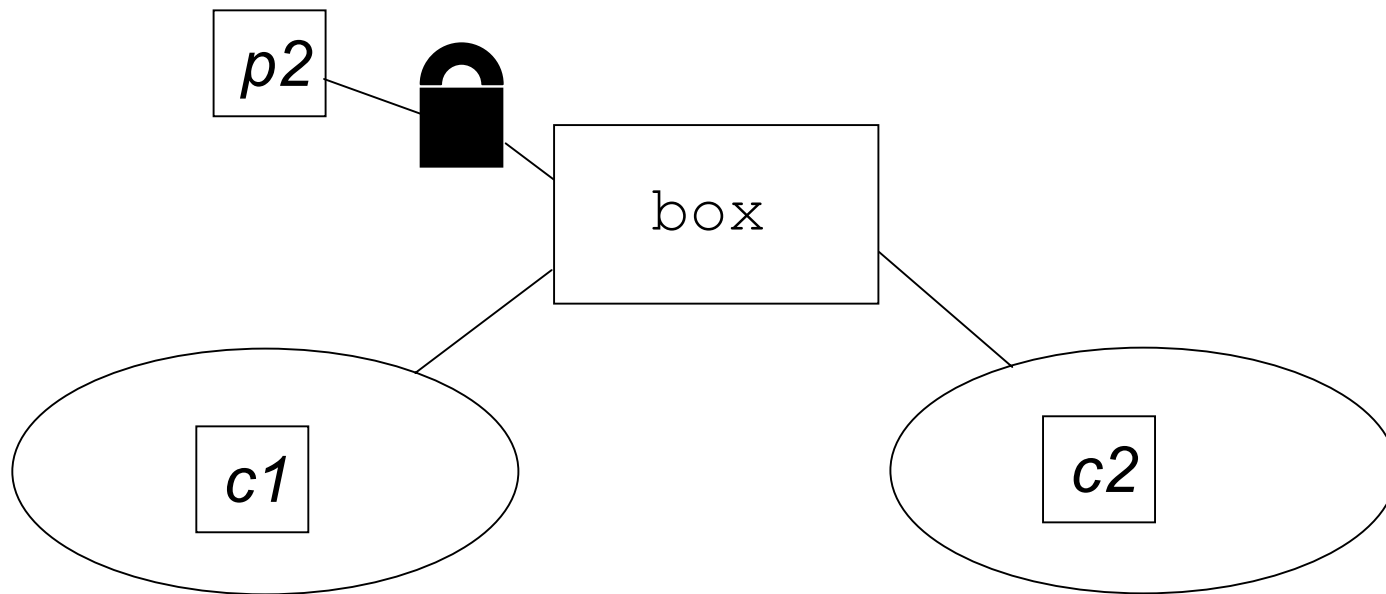
Producer/Consumer Execution using notify()

- *p1* assigns `false` to `empty`, then invokes `notify()`
 - *c1* is reenabled for thread scheduling (could have been *c2*)



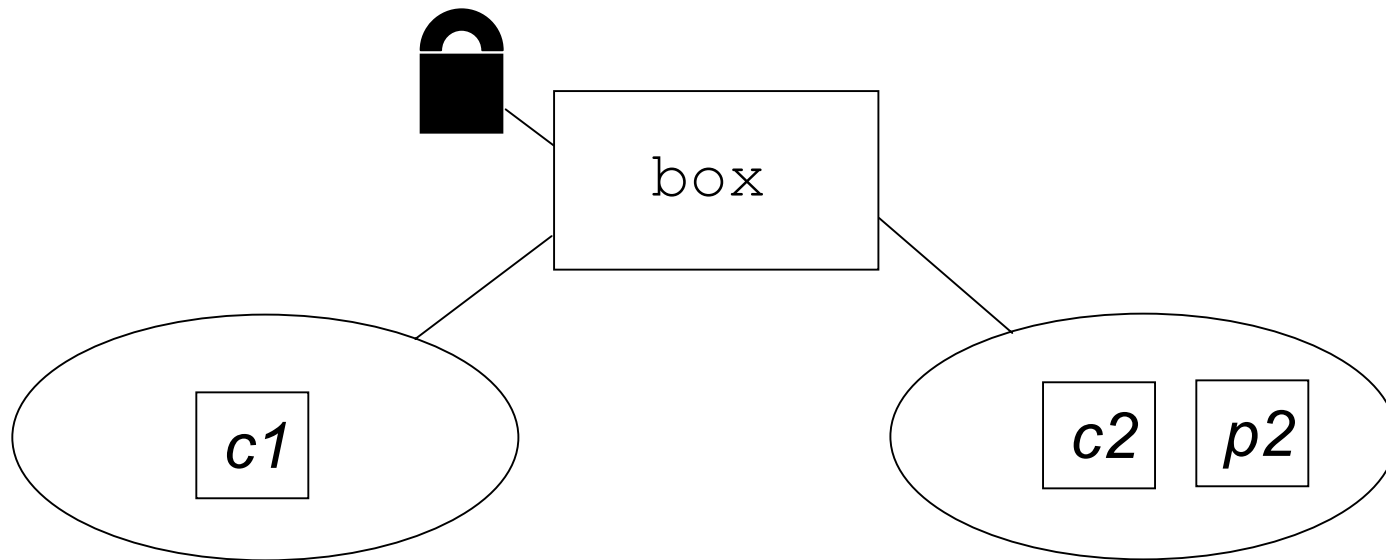
Producer/Consumer Execution using notify()

- *p1* returns from `put()` and relinquishes the lock
- The JVM grants the lock to *p2* (it could have granted it to *c1*), which starts executing `put()`



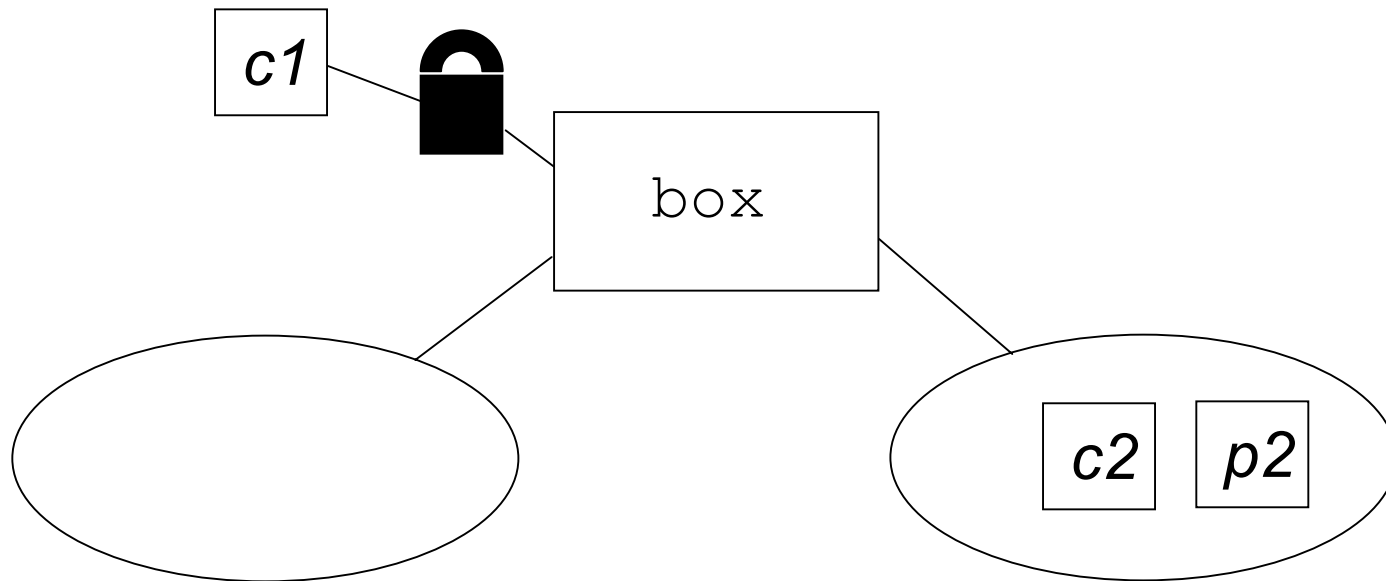
Producer/Consumer Execution using notify()

- *p2* enters the `while` loop body (`empty` is `false`)
 - it invokes `wait()`, releases the object's lock, is placed in the wait set and blocks



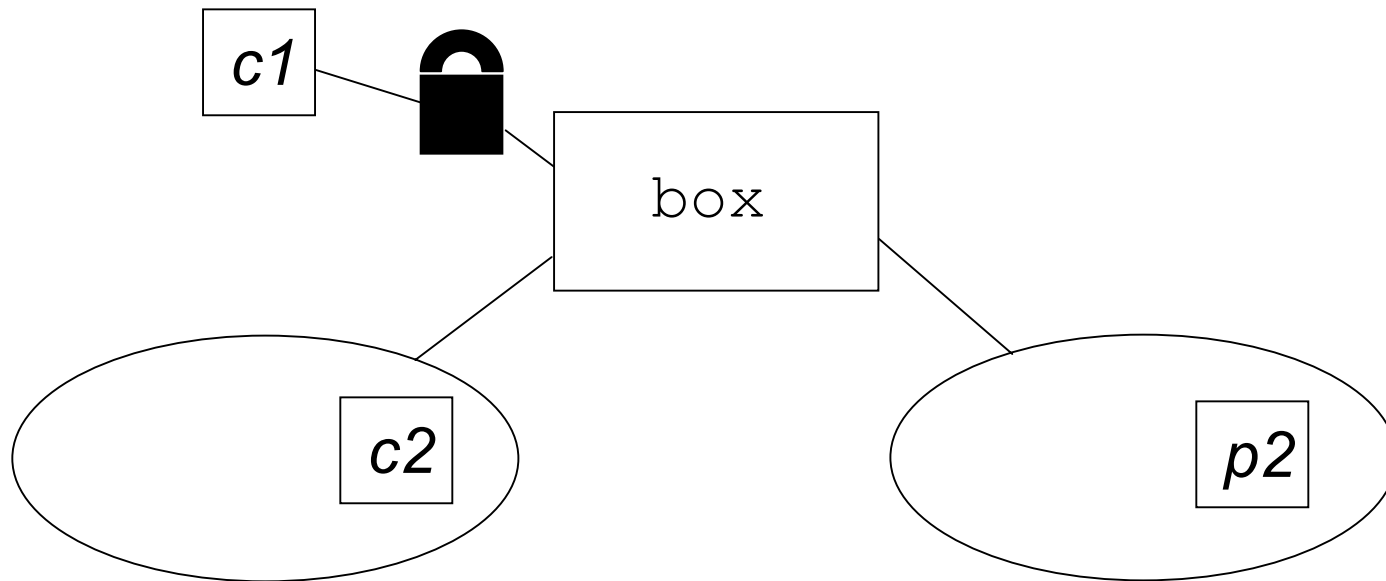
Producer/Consumer Execution using notify()

- The JVM grants the lock to *c1*, which returns from `wait()`



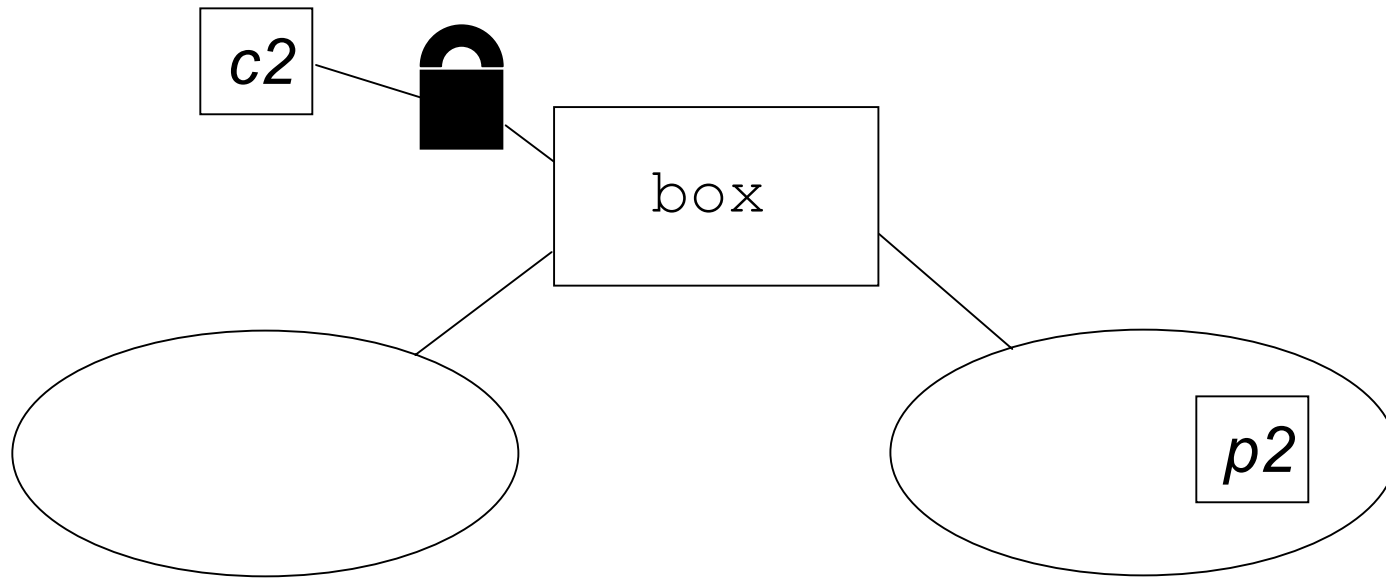
Producer/Consumer Execution using notify()

- *c1* exits the `while` loop (`empty` is `false`), assigns `true` to `empty` and invokes `notify()`
 - *c2* is reenabled for thread scheduling (could have been *p2*)



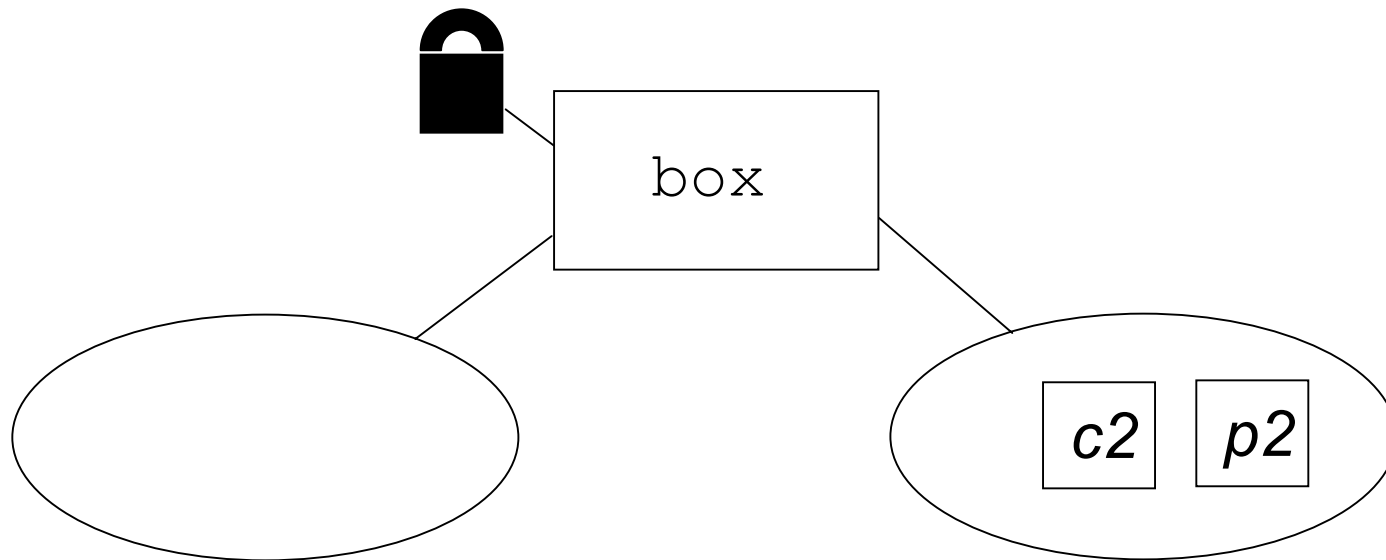
Producer/Consumer Execution using notify()

- *c1* returns from `get()` and relinquishes the lock
- The JVM grants the lock to *c2*, which returns from `wait()`



Producer/Consumer Execution using notify()

- `c2` enters the `while` loop body (`empty` is `true`)
 - it invokes `wait()`, releases the object's lock, is placed in the wait set and blocks



Producer/Consumer Execution using notify()

- we're now stuck until another producer comes along

